UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TRABAJO DE GRADO

WMVC FRAMEWORK

UN FRAMEWORK MVC PARA EL DESARROLLO

DE APLICACIONES WEB Y MOBILE EN .NET

FERNANDO ARÁMBURU

Director: Dr. Gustavo Rossi

La Plata 2005

Agradecimientos

- Principalmente a mi hermano. Por apoyarme incondicionalmente, por enseñarme muchísimas cosas tanto de la facultad como de la vida y por ayudarme siempre, esté donde esté.

- A mis padres, por brindarme continuamente el apoyo necesario para seguir adelante y por bancarme durante todo este tiempo.

- A mi familia entera por ayudar cada uno a su manera.

- A mis amigos, Guille, Pablo, Leo y Matías, por aguantarme así como soy y por ayudarme a llegar hasta acá.

- A mis compañeros de estudio de toda la carrera y amigos, Enrique, Matías y "el Chola" por permitirme encontrar en un compañero de estudio algo más que eso: un amigo.

- A mis amigos en general, porque aunque no aparezcan sus nombres aquí, ayudaron de una forma u otra a que yo finalice este trabajo.

- A Gustavo, mi director, por aguantar mis idas y vueltas con este trabajo y conseguir con su experiencia guiarme durante todo este tiempo.

- A mis compañeros de trabajo, por motivarme a estudiar e insistirme siempre en que cada día me encontraba mas cerca del final del proyecto.

- A todos los que brindaron ayuda técnica frente a las dudas y problemas que me encontré durante el trabajo.

- A todos los que forman parte del proyecto de Cerveza Artesanal: a mis amigos Pablo y Guille por pasar días enteros con ellos trabajando para esto y a Catty y Jorge por aguantarnos en su casa durante todo este tiempo.

Índice

Introducción

I. Motivación

El desarrollo de aplicaciones web ha sido siempre un tema de estudio e investigación debido, no solo a su complejidad, sino también al enorme mantenimiento que cualquier aplicación de este tipo puede llevar.

Las primeras aplicaciones web eran desorganizadas y muy difíciles, sino imposibles, de mantener y extender. En ellas, las páginas se comunicaban directamente con el modelo, para hacerle un requerimiento, teniendo como consecuencia directa que, ante una pequeña modificación en el modelo, la mayoría de la aplicación web debía también modificarse para adaptarse a los cambios realizados.

Con el tiempo se desarrollaron grandes avances en este campo con la intención de separar el modelo de la vista. La aparición del MVC (Model - View – Controller) brindó un marco teórico importante, para el desarrollo de frameworks basados en él. Un ejemplo exitoso de implementación MVC, es la arquitectura de JSP/Servlet de Java, que permite la comunicación entre aplicación web y servlets, y de éstos con el modelo, evitando que sea la aplicación web la que interactúe con el modelo de la aplicación.

Sin embargo, generalmente, los desarrolladores web, terminaban escribiendo código de Java en las páginas JSP para poder mostrar los datos que los servlets habían obtenido. La arquitectura de JSP/Servlet tenía sin embargo, un problema muy importante: las páginas web debían contener código Java. Así, un diseñador de páginas web no podía trabajar con la arquitectura o debía conocer Java para poder hacerlo, llevando esto a páginas con menor contenido gráfico.

La aparición de Jakarta Struts, otra implementación más moderna de MVC, permitió mejorar notablemente el desarrollo y diseño de las aplicaciones web. Struts separó el trabajo de un diseñador y de un desarrollador: ahora, cada uno podía trabajar independientemente del otro pudiendo aprovechar al máximo sus capacidades. El único problema que posee la comunidad de desarrolladores de aplicaciones web es que actualmente solo Struts soporta una separación completa

entre el desarrollo y el diseño permitiendo que en las páginas web no existan ni una sola línea con código Java.

Por otra parte, una vez que las aplicaciones web eran desarrolladas fácilmente por desarrolladores Java utilizando frameworks MVC bajo el paradigma de orientación a objetos, Microsoft lanzó al mercado el .NET Framework. Este framework provee varios lenguajes orientados a objetos además de la plataforma ASP.NET, la cual fue creada para desarrollar aplicaciones web utilizando el .NET Framework.

Sin embargo, y a pesar de todo el esfuerzo realizado por Microsoft, el .NET es un framework relativamente nuevo y por lo tanto, por mas que existan algunos frameworks basados en MVC, no existe aun uno que realmente satisfaga las necesidades de mejorar y facilitar el desarrollo de aplicaciones web, mobile y de escritorio. Debido a ello, los grupos de desarrolladores de aplicaciones web y mobile que utilizan el .NET Framework (y por lo tanto ASP.NET) para el desarrollo de sus aplicaciones, corren con la desventaja de no poseer un framework consistente que los ayude a realizar sus tareas de una manera mas organizada, efectiva, veloz y cómoda.

II. Objetivo

El objetivo del trabajo es desarrollar un framework basado en la arquitectura MVC (Model-View-Controller), que permita a los desarrolladores de aplicaciones web y mobile una posibilidad más organizada, veloz y efectiva de crear y mantener sus aplicaciones en el ambiente de .NET.

Para lograr esto, primero analizaremos las características principales de MVC y el modelo de programación de aplicaciones ASP.NET para luego, diseñar y desarrollar un nuevo modelo que permita la aplicación de la arquitectura MVC en los sistemas móviles y web desarrollados con .NET Framework.

III. Contribuciones del trabajo

El framework provee una buena implementación de MVC y logra separar el modelo de la vista contribuyendo al ambiente .NET y a su grupo de desarrolladores con un framework completo, simple, fácil de usar y extensible para el desarrollo de aplicaciones web y mobile. Este mismo fue

basado tanto en ideas obtenidas de experiencias anteriores, como de ideas buenas provistas por los frameworks ya existentes en el mercado así como también del conocimiento e investigación a muy bajo nivel de ciertas características, algunas casi inexploradas, del propio .NET Framework.

A su vez este framework, si bien no es un desarrollo que se encuentre preparado para poner en producción en un sistema de software real, dado que no se le ha realizado un testeo exhaustivo sobre fiabilidad, estabilidad y performance, aporta una base sólida y permite que con pequeños cambios su creación como producto de software sea posible.

Por último, el desarrollo y la documentación provista por este trabajo, dejan abierta la posibilidad de que otras vistas diferentes de web y mobile, o éstas mismas pero con implementaciones más especializadas, utilicen este framework mediante el desarrollo de addins fáciles de diseñar y codificar.

V. Guía sobre los capítulos del trabajo

El trabajo se encuentra dividido en 2 partes a modo general para que luego, cada una de esas partes se encuentre dividida en varios capítulos. A continuación, haremos un breve barrido sobre ambas partes del trabajo mientras que iremos introduciendo cada capitulo listando los temas que trata cada uno de ellos.

La primera parte, describe en forma general, todas las herramientas bases que le brindan soporte al Framework WMVC así como también, un poco la historia del porque, tanto en el desarrollo de aplicaciones como en la web, las cambiantes necesidades del software llevaron a una evolución y cambio radical en la forma de construir software.

Dentro de esta primera parte, en el primer capítulo se realiza un breve recorrido por el desarrollo de aplicaciones web, desde los primeros tiempos con la utilización de tecnologías como CGI hasta las tecnologías actuales como JSP/Servlets.

En el capítulo 2, se presenta el MVC, incluyendo las ventajas y desventajas de aplicar el mismo en el desarrollo de aplicaciones. A su vez, este capítulo también contiene un reducido donde

se avanza un poco sobre la evolución del mismo, y sobre las implementaciones de frameworks o productos basados en esta arquitectura.

Como cerrando la primera parte del trabajo el .NET Framework y más específicamente ASP.NET, son presentados en el capítulo 3.

La segunda parte del trabajo describe en forma completa el objetivo del trabajo: el WMVC Framework, desde sus funciones generales a su diseño e implementación de cada uno de los componentes que lo conforman.

Durante el capítulo 4 se presenta al Framework WMVC en forma general. Se muestra una división en componentes del mismo y se hace una breve introducción a cada uno de los mismos, así como también se hace la descripción de algunos conceptos propios del Framework. Sobre el final, se enumeran en forma muy general los pasos que involucra la ejecución de una acción WMVC.

En el capítulo 5, se efectúa una descripción más detallada de cómo realmente el desarrollador debe comunicarse con el Framework. Se introducen ciertos conceptos más avanzados y exclusivos con el objetivo de que el lector comprenda ciertas características y advierta sobre las ventajas que el uso de este Framework provee. Para finalizar el capitulo, se realiza en un nivel bastante mas detallado, una descripción de las etapas que atraviesa una acción.

El componente más importante del Framework, el “Framework Core” es analizado en el capítulo 6, realizando una división en módulos y detallando a nivel de diseño e implementación los distintos componentes de los mismos.

Durante el capítulo 7 se presenta la biblioteca de controles. Es aquí donde se introducen los distintos tipos de controles provistos por el framework para interactuar con la vista, sino que también se mencionan las ventajas de la utilización de éstos frente a los provistos por el .NET Framework.

En el capítulo 8 se hace una introducción a los componentes extras que facilitan aun más la tarea de desarrollar aplicaciones con éste Framework.

En el capítulo 9 se enumeran las conclusiones del trabajo y se presentan los trabajos futuros que este proyecto deja abierto.

Finalmente, el Apéndice A presentada un informe completo acerca de la configuración del WMVC Framework junto a una breve ejemplo sobre el cual se muestran las ventajas de dicha forma de configuración.

Primera Parte

Capítulo 1: Evolución del desarrollo de las aplicaciones Web

1.1 Breve Introducción

En los comienzos, las aplicaciones web consistían básicamente en documentos HTML con pequeñas partes de código CGI incluidos únicamente para manejar y procesar los envíos de formularios HTML. La programación de dichas aplicaciones consistía, generalizando, en la creación de páginas HTML para luego crear unos pocos scripts de CGI para manejar en forma específica los formularios <form>.

Apenas se comenzó a difundir el uso de Internet, se volvió obvio que el enfoque de desarrollo de aplicaciones que había en ese momento era demasiado restrictivo dado que los documentos HTML eran estáticos y no podían reflejar información actualizada. Complementando el panorama, los desarrolladores tenían la responsabilidad de mantener la correspondencia entre los formularios de las páginas HTML con el procesamiento CGI, lo que no solo no era fácil sino por demás complejo.

Los Servlets y tecnologías similares resolvieron el problema mediante la generación de cada página en forma programática. Básicamente, un método o una serie de ellos eran escritos en el lenguaje de programación que el desarrollador eligiera para generar todas las páginas de la aplicación. De esta manera, las aplicaciones podían ser tan potentes como una aplicación de escritorio teniendo todo su contenido en un conjunto de archivos de código fuente del lenguaje elegido, logrando así la facilidad de mantener la correspondencia entre el HTML generado y el procesamiento del mismo. Con este tipo de desarrollo, las aplicaciones web entraban perfecto en lo que en ese momento era el desarrollo tradicional de software: generar, a partir de las especificaciones, un software que cumpla con las especificaciones.

Sin embargo, tan pronto se comenzaron a modificar dichas aplicaciones, los desarrolladores notaron que si se deseaba cambiar algo muy simple en la vista, como podría llegar a ser la falta de un tag </table> o la modificación de la navegación del sitio, se debía ir y modificar el código fuente, convirtiendo a las modificaciones en un trabajo sumamente tedioso. Por otro lado también apreciaron que esa forma de desarrollar aplicaciones no se correspondía con la estructura de links que necesitaban tener las aplicaciones web.

Allí fue cuando aparecieron las páginas de servidor o "Server Pages" con el objetivo de solucionar mediante la creación de páginas HTML conteniendo no solo los tags normales de cualquier página HTML sino también pedazos de código embebido que el servidor interpreta dinámicamente.

Con esta tecnología, los desarrolladores que conocieran tanto la sintaxis de HTML como el lenguaje embebido podían mezclar ambas para obtener el mejor provecho. Así, las aplicaciones podían desarrollarse con HTML estático para la creación de la vista y luego aumentada o completada con el código embebido para proveer las funciones dinámicas necesarias. Las Server Pages trabajaban razonablemente bien asumiendo que los desarrolladores tenían la capacidad de escribir tanto el código del lenguaje de programación como el lenguaje HTML

1.2 CGI

Tal cual mencionamos en "1.1 Breve introducción", CGI fue la primera herramienta que permitió a las aplicaciones web tener algo de dinamismo. Con CGI (Common Gateway Interface) [3] todas las acciones eran efectuadas por programas llamados Scripts, que eran levantados por el servidor web en respuesta a una solicitud de una URL específica. Una vez iniciado, el script ejecutaba todo el procesamiento necesario, generaba una respuesta y enviaba ésta nuevamente al navegador del cliente [1].

La ventaja que poseían los programas CGI, era que podían ser codificados en cualquier lenguaje que cumpliera con mínimos requisitos (como poder leer el STDIN y escribir el STDOUT) y así escapar del mundo estático de HTML [4].

Sin embargo, el principal problema que poseía CGI era que separaba totalmente el desarrollo de la aplicación web en dos partes: por un lado, los desarrolladores web que escribían HTML y algún script en el cliente y por otro, los programadores de componentes CGI que corrían en el server. Lo malo de esta división era que en realidad esa división no era total porque el desarrollador web debía conocer y llamar a los scripts CGI tal cual habían sido definidos y los programadores CGI necesitaban manejar cualquier posible entrada trayendo esto serios problemas [2] [4].

1.3 La aparición de PHP

PHP es un lenguaje de script del lado del servidor que trabaja dentro de un documento HTML para producir contenido dinámico. Justamente, la mayor ventaja de de esta tecnología fue que, con el uso de unos tags especiales que indican la existencia de código PHP, trabajaba dentro mismo del documento HTML, por lo que no era necesario escribir una rutina en otros lenguajes como C o Perl con muchos comandos generando la salida [7].

```
<html>
    <head>
        <title>Today Example</title>
    </head>
    <body>
        <?php
            $today = getdate();
            echo $today;
        ?>
    </body>
</html>
```

Figura 1: Ejemplo en PHP que muestra la fecha actual en una página web

Lo que llevó a PHP a ser una de las tecnologías más usadas en el desarrollo de aplicaciones web y que dejó completamente obsoleta a su predecesor fue que con PHP uno podía escribir en dos o tres líneas lo que en CGI llevaba muchas logrando así:

- Facilidad para desarrollar aplicaciones
- Mayor velocidad al momento de la ejecución.
- Un mantenimiento no tan complicado.

PHP en su primera versión, apareció cuando un desarrollador llamado Rasmus Lerdorf programó un script en Perl/CGI que le permitía saber cuantos visitantes habían visitado su sitio web y a su vez mostrar esa cantidad. Para ese entonces esa funcionalidad era completamente novedosa por lo que Lerdorf y un grupo de desarrolladores decidieron continuar con esos avances. Un tiempo más tarde, ese grupo consiguió generar una rutina en C que permitía convertir todos los datos de un formulario web en variables permitiendo así exportarlas o usarlas en otros lenguajes [5].

Así continuaron hasta 1997 cuando salió la versión 2.0 de PHP totalmente escrita en C, que permitía embeber código PHP dentro de la página HTML [6]. Las versiones posteriores de PHP fueron mejorando a las anteriores y agregándole funcionalidad como por ejemplo:

- La creación de variables globales como Session, Request y Response.

- La posibilidad de acceder a distintas Bases de Datos
- El manejo de otros tipos de archivos como imágenes, pdfs.

1.4 Microsoft y su ASP

Active Server Pages es un ambiente de desarrollo en el cual, de la misma forma que PHP, uno puede combinar HTML y scripts además de poder reusar componentes ActiveX para crear soluciones de negocios mediante aplicaciones web. ASP, soporta tanto VBScript como JScript y corre generalmente sobre el servidor web de Microsoft: Internet Information Server (IIS) [8].

Con ASP, el desarrollador crea archivos con extensión .asp, el cual puede contener una combinación entre HTML, scripts del lado del cliente como puede ser javascript o Visual Basic Script y componentes a ejecutar del lado del servidor escritos en cualquier lenguaje.

Para entender mejor el funcionamiento de esta tecnología, pongamos como ejemplo un formulario que es utilizado para enviar en una solicitud el símbolo de la acción de bolsa que se desea comprar. Cunado esa información es recibida por el servidor, la página ASP receptora utiliza en sus primeras líneas al componente que habla con servidor de precios de acciones de bolsa de manera de obtener el precio. Una vez allí, mediante el uso de HTML se puede mostrar el precio buscado. Programando de esta manera, se logra que el desarrollador se enfoque sobre como comunicarse con el servidor de precios de acciones mientras que simultáneamente el autor de páginas web se focaliza en la vista y sus componentes sin prestar atención al como hablar con el servidor [9].

ASP, al igual que PHP, utiliza delimitadores para encerrar secuencia de comandos a ejecutar en el servidor mediante los tags "<%" y "%>". Dentro de esos tags se pueden asignar valores a variables, obtener información del cliente y/o utilizar dicha información para hacer algún procesamiento en el servidor. El lenguaje de script provisto por ASP también posee sentencias de control de flujo como if y while que permiten darle cierta lógica que a veces la vista necesita para obtener o mostrar los datos correspondientes [9].

```
<HTML>
    <HEAD>
        <TITLE> Date Example </TITLE>
    </HEAD>
    <BODY>
        <FONT COLOR="GREEN">
        <%
                Response.Cookies("Now")=Time;
                If Time >= #12:00:00 AM# And
                   Time < #12:00:00 PM# Then
        %>
                        Good Morning!
                <%Else%>
                        Hello!
                <%End If%>
        </FONT>
    </BODY>
</HTML>
```

Figura 2: Ejemplo en ASP que muestra la fecha actual en una página web

Por ejemplo, en el ejemplo de la figura 2 que se muestra inmediatamente arriba se puede apreciar una página ASP que obtiene la fecha actual del servidor y la guarda en una cookie en el cliente además de cambiar el mensaje de bienvenida según la hora.

1.5 La primera propuesta de Java: Servlets

En 1998, Sun Microsystems lanzó la primera versión de Java Servlets [10], la cual fue altamente aceptada como lenguaje de programación para aplicaciones web dado que Java traía con ello escalabilidad, productividad y flexibilidad. A diferencia de otros lenguajes, esta tecnología contaba con algunas ventajas como:

- Los programas en Java son más fáciles de desarrollar y testar.
- Java es un lenguaje orientado a objetos
- Java es independiente de la plataforma, del servidor web
- Java es un lenguaje existente por lo que permitía programar el modelo de las aplicaciones de la misma forma que en aplicaciones de escritorio, teniendo eso como ventaja la posibilidad de usar ilimitada cantidad de componentes como seguridad, logueo y acceso a base de datos vía estándares como JBDC.

```
<FORM METHOD="GET"
      ACTION="/servlet/ViewAppointmentServlet">
  Enter your Name:
  <INPUT TYPE="text" NAME="username"/><BR>
  <INPUT TYPE="submit" VALUE="Search!"/>
</FORM>
```

Figura 1.3: Ejemplo de un formulario web llamando a un Servlet

Un servlet típico consulta al modelo por información, la cual es luego enviada en formato HTML al cliente. Aquí presentamos un ejemplo en el cual mediante una página con el formulario web de la figura 1.3 se llama a un servlet (cuyo código fuente se encuentra en la figura 1.4) el cual busca y genera la salida adecuada.

```
public class ViewAppointmentServlet extends HttpServlet {

  public void doGet(HttpServletRequest req, HttpServletResponse resp)
              throws ServletException, IOException {

     // determine the parameters from the HTTP request.
     String userName = req.getParameter("username");

     // open output-stream to client for response.
     resp.setContentType("text/html");
     PrintWriter out = new PrintWriter(resp.getOutputStream());

     ResultSet rs = model.getAppointment(username);

     // create an HTML table.
     out.println("<HTML>");
     out.println("<HEAD><TITLE>Appointment Schedule</TITLE></HEAD>");
     out.println("<BODY>");
     out.println("<TABLE>");
     out.println("<TR>");
     out.println("<TD>Date</TD>");
     out.println("<TD>Start Time</TD>");
     out.println("<TD>End Time</TD>");
     out.println("<TD>Name</TD>");
     out.println("</TR>");

     // iterate through the ResultSet.
     while (rs.next()) {
       out.println("<TR>");
       out.println("<TD>" + rs.getString("Date") + "</TD>");
       out.println("<TD>" + rs.getString("StartTime") + "</TD>");
       out.println("<TD>" + rs.getString("EndTime") + "</TD>");
       out.println("<TD>" + rs.getString("Name") + "</TD>");
       out.println("</TR>");
     }
     out.println("</TABLE></BODY></HTML>");
  }
}
```

Figura 1.4: Código fuente de un Servlet

Sin embargo, aunque con la utilización de los Java Servlets [10] uno tiene enormes posibilidades de desarrollo, también tenía desventajas importantes:

- Era necesario una gran cantidad de código para realizar tareas simples.
- Al ser todo código Java, una modificación en la presentación necesitaba de una nueva compilación del servlet
- Modificaciones en la vista necesitaban ser realizadas por gente con conocimientos de Java ya que toda la salida del programa era generada por el lenguaje.
- Código poco claro debido a la existencia de muchas líneas dedicadas a llamar a métodos visuales como `out.PrintLn()`.

1.6 La llegada de Java Server Pages

El diseño original de los servlets necesitaba que cualquier modificación sea realizada sobre el código fuente escritos en Java asumiendo que todos los participantes del proyecto tenían conocimientos del lenguaje, lo cual no era en la práctica algo usual. Fue entonces cuando Sun lanzó JSP [11], un nuevo producto suplementario a la API de Servlets.

JSP permitía a los desarrolladores, en lugar de encapsular la presentación dentro del código Java, como típicamente se hacía en los servlets, escribir HTML directamente y encapsular el código Java dentro del HTML mediante el uso de tags especiales (<% %>), tal cual lo hacían ASP y PHP. De esta manera, en tiempo de ejecución, el compilador evaluaba la página JSP y generaba a partir de ella un servlet [11].

```
public class AppointmentBean : Serializable
{
    // only accessible through accessor and mutator methods.
    private String userName = "";

    public AppointmentBean() {}  // a no argument constructor

    // Required accessor and mutator methods for the userName property
    public String getUserName()  {return userName;}
    public void setUserName(String sz)  {userName = sz;}

    // Finds appointments for a particular user from a database
    public ResultSet executeQuery()
    {
        // the ResultSet is sent back to the JSP page context.
        return model.getAppointments(this.userName);
    }
}
```

Figura 1.5: Java Bean para el mismo ejemplo presentado con Java Servlets

La gran ventaja de JSP era que permitía separar la presentación de la lógica de negocios, logrando por primera vez que los arquitectos e ingenieros de aplicaciones web puedan construir frameworks independientemente de la lógica de presentación. Así, JSP conseguía que los diseñadores gráficos y los programadores java trabajen sobre archivos diferentes aprovechando cada uno al máximo sus posibilidades. Las pequeñas partes de código java que se debían introducir en las páginas se realizaban mediante el uso de JavaBeans, una tecnología de java que permitía a los desarrolladores encapsular la mayoría de la lógica de negocio necesitada en la página web.

Para explicar mejor, volvemos sobre el ejemplo anterior pero ahora presentado con JSP y JavaBeans. Arriba, en la figura 1.5 se puede ver el bean AppointmentBean teniendo un solo método, `executeQuery()`, en el cual se llama al modelo y retorna un ResultSet al llamador del

mismo sin tener que escribir dentro del Bean ni una sola línea de código de presentación como lo habíamos visto para el caso de Servlets.

```
<BODY>
  <%@ page language="java"
        import="java.sql.*,AppointmentBean" %>
  <jsp:useBean id="apptbean" scope="session"
      class="AppointmentBean" />
  <H3>Today's Appointments:</H3><BR>
  <B>The following are your appointments:</B><BR>

  <TABLE>
    <TR>
      <TD><B>Date:</B></TD>
      <TD><B>Start Time:</B></TD>
      <TD><B>End Time:</B></TD>
      <TD><B>Patient Name:</B></TD>
    </TR>

    <!-- Get the appointment information -->
    <%
      ResultSet rs;
      apptbean.setUserName(request.getParameter("username"));
      rs = apptbean.executeQuery();

      while (rs.next())
      {
    %>

    <TR>
      <TD><B> <%= rs.getString("Date") %> </B></TD>
      <TD><B> <%= rs.getString("StartTime") %> </B></TD>
      <TD><B> <%= rs.getString("EndTime") %> </B></TD>
      <TD><B> <%= rs.getString("PatientName") %> </B></TD>
    </TR>

    <%
      }
      rs.close();
    %>

  </TABLE>
</BODY>
```

Figura 1.6: HTML el mismo ejemplo presentado con Java Servlets

La figura 1.6 representa el código de la página JSP el cual contiene la estructura típica de una página HTML con el agregado de código embebido java como se puede ver tanto para llamar al bean para que ejecute el query como para iterar sobre la colección de resultados y generar la tabla.

Sin embargo, la primera versión de JSP seguía teniendo una desventaja en la presentación. Las páginas JSP tenían demasiado código java generando aplicaciones medianamente difíciles de debaguear y mantener. Así fue entonces como a finales de 1999 se lanzó la versión 1.1 de JSP que introducía la idea de: "Custom Tags" o tags personalizados que a pesar de ser nueva para java, no lo era para todos los desarrolladores ya que estos Custom Tags existían desde 1995 con Cold Fusion [12].

Los tags personalizados [12] resolvieron un problema fundamental de la tecnología dando a las páginas JSP un look and feel familiar para cualquier diseñador Web debido a que, como se ve en la figura 1.7 no era necesario escribir código fuente para por ejemplo iterar sobre una colección de resultados obtenidos desde un javaBean.

```
<customtag:foreach group="x">
<TR>
    <TD> <%= x.get("Date") %> </TD>
    <TD> <%= x.get("StartTime") %> </TD>
    <TD> <%= x.get("EndTime") %> </TD>
    <TD> <%= x.get("Name") %> </TD>
</TR>
</customtag:foreach>
```

Figura 1.7: Ejemplo de un custom tag para iterar una conjunto de elementos

Los Custom Tags eran simplemente abstracciones de funcionalidad java definidos mediante un lenguaje de tags haciéndolos parecer a los propios tags de HTML, que permitían un uso más natural de dicha funcionalidad. Internamente, los tags personalizados eran desarrollados por desarrolladores JSP y durante la ejecución, el compilador al encontrar uno de estos tags llama al manejador de tag adecuado mediante los mensajes doStartTag y doEndTag [12].

Capítulo 2: Arquitectura MVC para la web

2.1 Introducción al MVC

En esta sección introduciremos el concepto de Model View Controller, describiendo sus componentes, funciones, ventajas y desventajas.

2.1.1 Necesidad del MVC

Las interfaces de usuarios son muy propensas a cambios a partir de requerimientos de los usuarios. Cuando uno extiende una aplicación, generalmente se debe modificar cierta funcionalidad correspondiente a la forma en que se debe mostrar como puede ser la modificación de un menú o la necesidad de portar el sistema a otra plataforma o sistema operativo, llevando, cualquiera de estas modificaciones a cambios en el código de la aplicación.

Aun más complicado es que diferentes usuarios, al momento de generar requerimientos de cambios en el sistema, estos sean conflictivos con funcionalidad existente o requerida. Esto sucede debido a las diferentes clases de usuarios existentes en el sistema, que llevan a que el sistema deba soportar de manera simple a un usuario avanzado que maneja la aplicación desde una consola mientras otro mas principiante la utiliza en una interfaz gráfica de ventanas.

La construcción de un sistema con dicha flexibilidad es algo caro y proclive a errores si es que la interfaz se encuentra ligada fuertemente al núcleo de la aplicación, o modelo de negocio. Para este caso, se necesitará desarrollar y mantener una serie de aplicaciones diferentes donde cada una posee el mismo modelo de negocio pero con interfaces distintas. Esto lleva como consecuencia que un cambio funcional necesite de modificaciones separadas en cada aplicación haciendo de dicho sistema algo complicado de mantener, controlar y extender.

Una vez explicado algunos problemas de sistemas con múltiples vistas necesitaríamos que:

- La misma información pueda ser presentada de diferentes formas.
- La presentación y el comportamiento de la aplicación reflejen los cambios hechos al mismo en forma inmediata.

- Cambios en la interfaz gráfica de la aplicación sean fáciles de realizar y hasta puedan ser realizables en tiempo de ejecución.
- Cambios en el "look and feel" del sistema no afecte al código del modelo de negocio de la aplicación.

2.1.2 Definición de MVC (Model – View – Controller)

El patrón de diseño Model - View – Controller (MVC) divide una aplicación interactiva en tres componentes:

- El modelo, el cual contiene la funcionalidad principal del negocio y los datos.
- La vista, encargada de mostrar la información a los usuarios.
- El controlador, cuya tarea es la de manejar y procesar la entrada del usuario.

El MVC, no solo provee el concepto de la división de responsabilidades entre componentes sino que también define los mecanismos necesarios para que la comunicación entre esos componentes se haga de la manera más efectiva y menos dependiente posible [13].

2.1.3 Estructura de MVC

El modelo contiene el núcleo funcional de la aplicación y los datos que él necesita. Además, el modelo, también exporta una serie de procedimientos los cuales ejecutan procesamiento específico de la aplicación. El controlador invoca estos procedimientos a partir de acciones del usuario. Por último, el modelo también provee funciones específicas para que las vistas puedan acceder a los datos que necesitan ser mostrados.

El componente vista es el encargado de presentar la información al usuario de la aplicación. Cada componente vista de la aplicación tiene la posibilidad de mostrar al modelo de una manera distinta. Por ejemplo, uno componente podría mostrar cierta información mediante un gráfico de torta mientras que otro similar puede estar mostrando la misma información pero vía una tabla de valores.

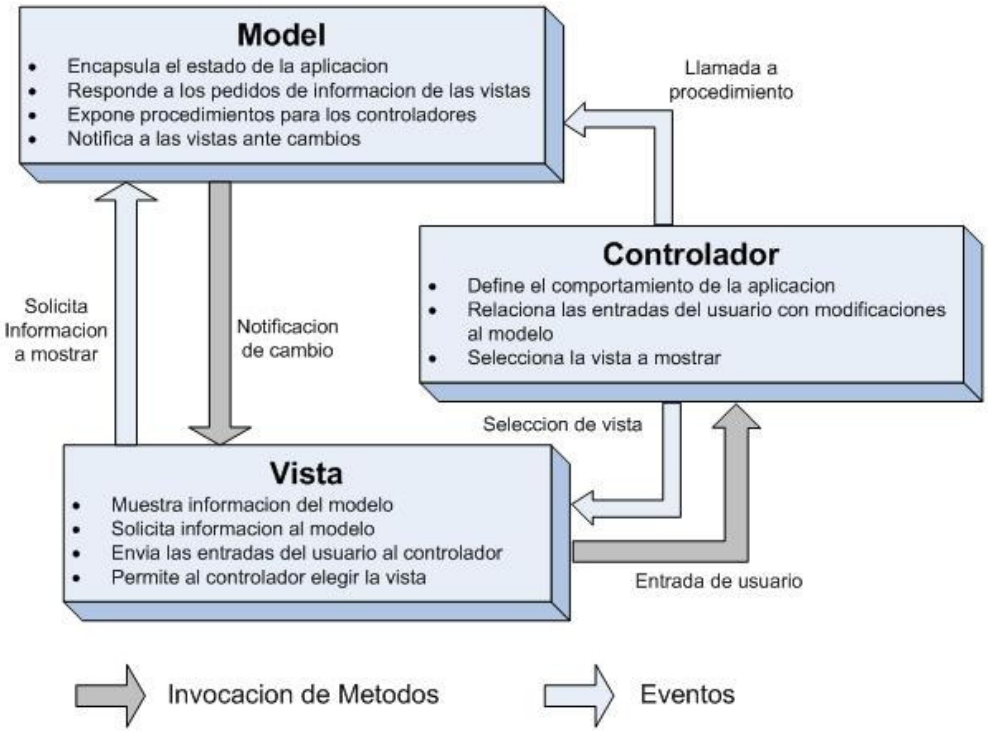


Figura 2.1: Componentes del MVC

El controlador del MVC es el encargado de aceptar y manejar los eventos del usuario y a su vez convertirlos en requerimientos para el modelo o la vista asociada al controlador [14]. La manera en que los controladores implementan el manejo de los eventos no es en este momento una cuestión importante por lo cual no forma parte del alcance del texto.

2.1.4 Funcionamiento del MVC

Una de las características principales de la arquitectura MVC es el mecanismo de propagación de cambios, el cual mantiene un registro de los componentes que dependen del modelo. Todas las vistas existentes (y según sea necesario también los controladores) registran sus necesidades de notificación ante cambios de manera que cuando el modelo cambie estas sean avisadas, siendo ésta la única relación entre el modelo y las vistas y/o controladores. A su vez, cada componente registrado define un procedimiento de actualización, el cual es activado por el mecanismo recién mencionado. Este procedimiento obtiene del modelo los valores que se deben mostrar y luego los hace visibles al usuario de la manera específica de cada vista [13].

Durante la inicialización del sistema, todas las vistas crean su respectivo controlador para luego registrarse al mecanismo de propagación de cambios del modelo, si es que fuera necesario.

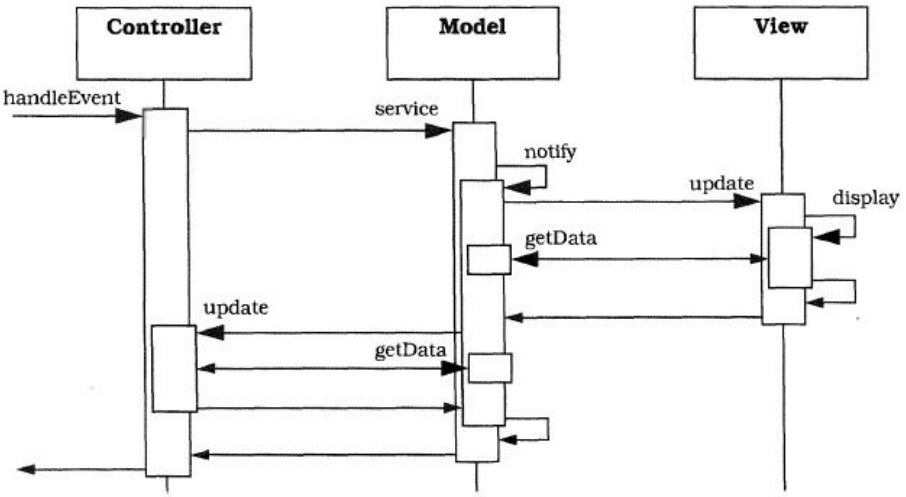


Figura 2.2: Funcionamiento del MVC

Para explicar un poco más al detalle el funcionamiento de la arquitectura enunciaremos aquí el conjunto de acciones que suceden en el momento en que el usuario genera un evento:

1. El controlador acepta la entrada del usuario mediante el manejo del evento, el cual es interpretado para luego activar un procedimiento específico en el modelo.
2. El modelo ejecuta el procedimiento requerido llevando éste a una modificación de su información.
3. El modelo, mediante el mecanismo de propagación de cambios, obtiene todas las vistas y controladores registrados al cambio ocurrido.
4. Por cada vista registrada, el modelo le notifica el cambio mediante el llamado a su procedimiento de actualización, opcionalmente llamado con algunos parámetros para especificar mejor lo ocurrido.
5. Cada vista solicita la información que necesita para luego actualizar su interfaz con los nuevos valores.
6. Cada controlador registrado obtiene información del modelo de manera de poder aplicar ciertos cambios en la funcionalidad accesible por el usuario.
7. Una vez finalizado todo el procedimiento de actualización de cada vista, el controlador original que detectó la entrada del usuario vuelve a tomar el control de la ejecución.

2.1.5 Consecuencias de MVC

La arquitectura MVC tiene, como toda arquitectura, ciertas ventajas que detallaremos brevemente [13].

- ***Múltiples vistas del mismo modelo***: MVC separa de forma estricta el modelo de los componentes vistas, lo que permite la construcción de múltiples interfaces visuales distintas para un mismo modelo de negocio.
- ***Sincronización entre vistas***: El mecanismo de propagación de cambios que posee el modelo asegura que todos los observadores (vistas y controladores) registrados sean notificados ante un cambio en la información manejada por el modelo en el tiempo real.
- ***Vistas y Controladores enchufables***: La separación de conceptos que trae el uso de MVC permite que uno cambie en cualquier momento la vista y los controladores del modelo de la aplicación. Esta flexibilidad permite la implementación de diferentes modelos de operación para la misma vista, tal como el de usuarios expertos versus principiantes o la construcción de vistas de solo lectura versus lectura/escritura dependiendo del usuario mediante el cambio únicamente del controlador.
- ***Cambios en el "Look and feel" del sistema***: Dado que el modelo es completamente independiente de la vista, la portación de una aplicación MVC a una nueva plataforma no debería afectar, de ninguna forma, el núcleo funcional del modelo.
- ***Posibilidad de construir frameworks basados en MVC***: Debido a su gran independencia y todas las ventajas que esta arquitectura brinda, se podría llegar a realizar frameworks que tengan como base al MVC

A su vez, ésta arquitectura también posee ciertas desventajas como:

- ***Aumento en la complejidad***: No en todos los casos la utilización de la arquitectura Model – View – Controller es la mejor manera de construir aplicaciones interactivas debido a toda la complejidad que introduce MVC en un sistema.
- ***Posible número excesivo de actualizaciones***: Para el caso que una acción del usuario resulte en múltiples actualizaciones, el modelo debería poseer la lógica para saltear notificaciones de cambios innecesarias evitando así una posiblemente excesiva e incontrolable cantidad de actualizaciones en las vistas y controladores.

- ***Relación fuerte entre la vista y el controlador***: La vista y el controlador en MVC, son componentes separados, pero generalmente fuertemente conectados privando así de su reuso en forma individual.
- ***Acoplamiento de vistas y controladores al modelo***: Debido a que tanto las vistas como los controladores hacen llamadas directas al modelo, cambios en éste último provocarían cambios tanto en las vistas como en los controladores asociados.
- ***Acceso ineficiente a los datos del modelo desde la vista***: Dependiendo de la interfaz pública que presente el modelo, una vista podría llegar a realizar múltiples pedidos de información para obtener los datos que ella necesita mostrar. Este problema se agrava aún más debido a que entre todos esos múltiples requerimientos es muy probable que se este solicitando información innecesaria o que ya se poseía anteriormente.
- ***Uso de MVC con las herramientas actuales de desarrollo***: La utilización de toolkits de herramientas de alto nivel para la construcción de interfaces de usuario puede quitar las posibilidades del uso de MVC. Actualmente, muchas de estas herramientas definen su propio flujo de control, manejan eventos internamente, interpretando éstos últimos y ofreciendo llamadas a procedimientos para cada tipo de actividad de usuario. Por lo tanto, la mayor parte de la funcionalidad de los controladores ya esta provista por los toolkits, y no es necesario implementar explícitamente los componentes de MVC.

2.2 Evolución del MVC

El MVC fue originalmente pensado, durante los años 80, para resolver el desarrollo de sistemas existentes durante aquellos tiempos. Con el pasar de los años, el grado de complejidad y tamaño de las aplicaciones GUI ha ido creciendo notablemente mostrando con este avance ciertas incompatibilidades en la aplicación de MVC.

Al momento de su aparición, el objetivo principal era básicamente promover la separación de las aplicaciones en dos partes: el código que tenía que ver con el modelo por un lado y el relacionado con su presentación por el otro. Esta división definida por MVC no especifica un lugar u objeto encargado para todas las responsabilidades que surgen en una aplicación real. La separación de tres capas definidas por MVC es una división teórica que no se implementa directamente.

2.2.1 Limitaciones del MVC original

Con el avance de la complejidad en las aplicaciones, quizás el problema que más impactó en MVC fue el de donde ubicar la lógica de evaluar el estado resultante luego de la ejecución de una operación en el modelo y a partir de ello la decisión de qué vista el usuario debía obtener.

Es natural ver que este comportamiento no debía incluirse en la vista, debido a que, según lo definido en el MVC original, ésta se encontraba únicamente encargada de presentar la información al usuario de la aplicación. También es claro que si el modelo se hiciera cargo de ésta responsabilidad entonces estaríamos introduciendo en el modelo una indeseable dependencia con las vistas, lo cual no es soportado por MVC. Por último, si el controlador era el encargado de esa decisión, entonces el flujo de control de la aplicación se encontraría esparcido entre los controladores de las distintas vistas, llevando esto a que el sistema sea difícilmente controlable, modificable y extensible. Fue allí cuando se detectó la presencia de ciertas tareas para las cuales MVC no había asignado un responsable, al menos entre los tres componentes inicialmente definidos.

2.2.2 Lógica de la aplicación y lógica de negocios

Dada la evolución de las aplicaciones y la imposibilidad de añadir la "lógica de la aplicación" a cualquiera de los tres componentes especificados en el MVC, fue necesario extender el MVC original con el agregado de un cuarto componente encargado de determinar el flujo de las vistas a partir de los resultados de la ejecución del modelo [14].

En este MVC de 4 componentes, tanto la vista como el modelo mantienen las mismas responsabilidades del MVC original mientras que lo que antes era el Controlador ahora se lo llama controlador de entrada. El nuevo componente del MVC, el controlador de la aplicación es el componente encargado de:

- Coordinar y controlar el flujo del sistema.
- Ser mediador entre el modelo y los demás componentes del sistema logrando una mayor independencia del modelo.
- Opcionalmente podría ser el encargado de notificar los cambios del modelo y poseer el modelo de referencias de los componentes dependientes.

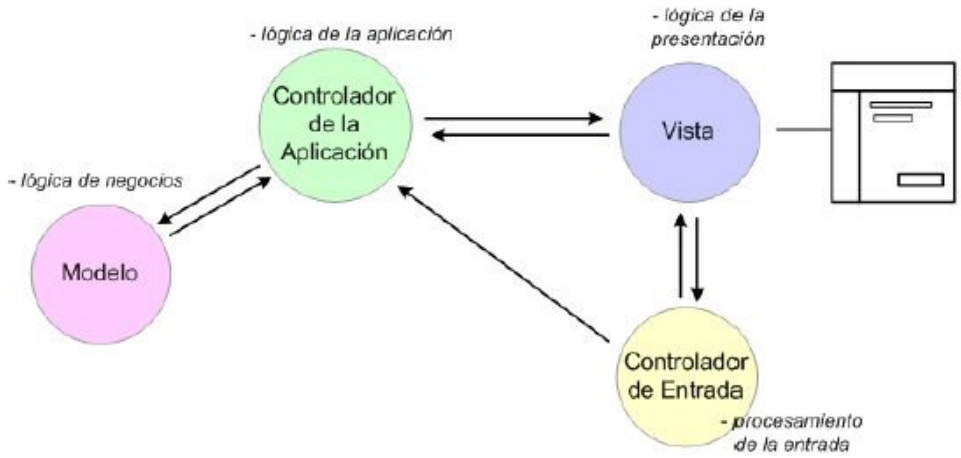


Figura 2.3: MVC actual con 4 componentes

2.3 MVC para aplicaciones Web

Originalmente, el MVC fue pensado para usarse en ambientes GUI tradicionales únicamente, pero, debido a la aparición de Internet y de las aplicaciones web y dado el éxito que habían conseguido las distintas implementaciones de MVC, se intentó migrar dicha arquitectura a la web. Sin embargo, para que una tecnología pueda implementar el MVC correctamente y obtener todas las ventajas de éste, ciertas características se deben cumplir que en el caso de la web no aplicaban como por ejemplo la inherente conexión sin estado entre el cliente y el servidor

2.3.1 MVC en la Web

Al igual que sucediera en el desarrollo de aplicaciones GUI, el aumento en el tamaño y en la complejidad de los sistemas provocó una necesaria renovación de las técnicas de diseño y construcción de las aplicaciones web. Con esa evolución, las aplicaciones web recorrían el mismo camino que las anteriores tecnologías, logrando dividir en componentes al sistema y definir responsabilidades para cada uno de ellos, entre las cuales se encontraba la separación de tareas entre los programadores y los diseñadores de interfaces.

Al igual que sucediera en el desarrollo de aplicaciones GUI, las aplicaciones web tuvieron su comienzo desordenado. Con el pasar de los años y el aumento en el tamaño y en la complejidad de las aplicaciones, el desarrollo tuvo grandes problemas debido a la falta de técnicas avanzadas de

construcción de software. Los sistemas no poseían una separación correcta de los distintos componentes lo que tenía como consecuencia directa software poco extensible y modificable.

La experiencia y los errores en éste área llevaron a una necesaria renovación de las técnicas de diseño y construcción de las aplicaciones web. Con esa evolución, las aplicaciones web recorrían el mismo camino que las anteriores tecnologías, logrando dividir en componentes al sistema y definiendo responsabilidades para cada uno de ellos, entre las cuales se encontraba la separación de tareas entre los programadores y los diseñadores de interfaces.

Fue en ese momento, cuando los desarrolladores notaron que la aplicación de Model - View – Controller en la web era posible y que el transporte de esa arquitectura a la web podía llegar a permitir una notable evolución en el desarrollo de la por aquel entonces nueva tecnología. Sin embargo, este pasaje del MVC fue, y hasta se podría llegar a decir que actualmente es, un gran problema debido a que existían factores inherentes a la web que imposibilitaban el pasaje del MVC original a esta tecnología como eran:

- Conexión sin estado entre cliente y servidor web.
- La vista de las aplicaciones web se encuentra en distinto lugar físico que el resto de los componentes.
- Grandes diferencias en las tecnologías utilizadas para la vista y la del resto de la aplicación.

2.3.2 Modificaciones sufridas por MVC en su transformación a la web

Debido a las características de la web contradictorias con el MVC explicadas en el punto anterior, el Model - View – Controller debió sufrir ciertas modificaciones para adaptarse al contexto de la web, algunas de las cuales explicaré brevemente a continuación [14].

Cambio en la responsabilidad del Controlador: En el MVC tradicional, el controlador tenía la responsabilidad de recibir un evento en forma de objeto y traducir éste en un llamado al modelo. Ahora, en el MVC Web, el controlador tiene como tarea la recepción del requerimiento http, procesar el mismo y convertirlo en un objeto evento de manera que él u otro objeto lo utilice para hacer un llamado al modelo.

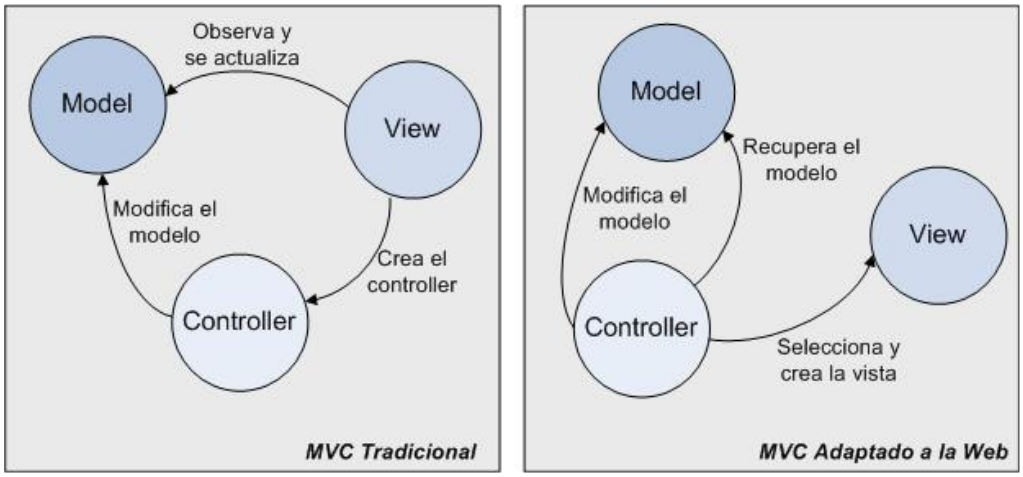


Figura 2.4: Cambio en las responsabilidades del controlador

Inversión en la creación de objetos vista – controlador: Debido a la ausencia de la vista como un objeto del lado del servidor, ahora el controlador, en lugar de ser creado por la vista, es él el encargado de crearla y enviarla al cliente, invirtiendo los roles de cada componente con respecto al MVC tradicional.

Construcción de la vista: En un MVC tradicional la vista es creada y ésta misma es la encargada de interactuar con el modelo para obtener la información que debe mostrar. Diferente completamente es lo que se sucede en las aplicaciones web, donde el controlador, simultáneamente al proceso de creación de la vista, debe poner disponible los objetos del modelo que la vista necesita para mostrarse. Este cambio se debe a que como la vista no se encuentra en el servidor, entonces no podrá, luego de ser enviada hacia el cliente, interactuar con el modelo para completar su contenido.

Mecanismo de notificación de cambios: Dado que la web no posee conexión para mantener el estado entre cliente y servidor, la vista se encuentra completamente desconectada del modelo, teniendo como consecuencia la imposibilidad de que un MVC Web implemente el mecanismo de notificación de cambios del MVC original. Algunos frameworks simulan esa conexión haciendo que la vista envíe periódicamente consultas al servidor web preguntando sobre si ha ocurrido algún cambio en la información que debe mostrar para en ese caso sí actualizarse y re enviarse hacia el cliente.

Posible inconsistencia en las vistas: En las aplicaciones web puede ocurrir una inconsistencia entre los datos del modelo y la información que se está mostrando en el cliente dado que la vista, al estar en un lugar físico distinto de los demás componentes de la aplicación y además encontrarse desconectada de ellos, no puede ser notificada ante un posible cambio del modelo.

Modelo y vista solo serán consistentes en el momento que el usuario envíe un requerimiento al servidor luego de que el mismo haya realizado alguna acción sobre la vista.

Transiciones entre vistas: Las transiciones que ocurren de una vista a otra no pueden ser directas en la web ya que ahora, el controlador debe intervenir para ello. Por otra parte, la intervención de un controlador tiene como consecuencia no solo una conexión indirecta entre las vistas sino además una interacción con el servidor, llevando ineficiencia y lentitud al proceso. En los MVC tradicionales para GUI, una vista podía abrirse como resultado de un evento en otra vista sin la intervención de ningún otro componente.

Eventos generados en la vista: En contraste a lo que ocurre en las aplicaciones de escritorio, en las aplicaciones web no se produce un feedback automático hacia el controlador a partir de un evento del usuario. Esto ocurre principalmente debido a que la vista y el resto de la aplicación residen en distintos lugares físicos. Un evento en la vista debe ser convertido en un requerimiento http lo que significa transformar toda la información existente en el cliente en texto, dado que el protocolo http solo transmite texto y no objetos ni eventos, para luego ser enviada y procesada por el controlador en el servidor. De esta forma se complica y ralentiza el procesamiento de un evento.

2.3.3 GUI MVC vs. Web MVC

Como mencioné recientemente, el patrón MVC fue desarrollado para ser utilizado en ambientes GUI y aunque la Web se podría llegar a decir que es una GUI (Graphical User Interface), ésta es muy diferente de un mundo de feedback instantáneo y orientado a eventos de usuario para el cual el MVC original fue pensado.

En un ambiente GUI tradicional, las acciones del usuario sobre la interfaz, por más pequeños que sean tienen la posibilidad de una reacción instantánea, como podría ser el habilitado/deshabilitado de un botón o de cualquier control. Mientras tanto, en un ambiente Web, un evento típico ocurre de una manera mucho mayor que como se había pensado originalmente en el MVC, debido a que un controlador web recién es notificado cuando o bien el usuario ha ingresado mucha información y quiere enviar ésta hacia el servidor o en el momento en que desea cambiar hacia una vista completamente diferente. En la web, cada uno de esos “eventos” definidos por el MVC original, no son tan detallados y sí mucho mas grandes en tamaño y más generales.

Diferencias entre uno y otro se presentan en todas sus partes. Aunque ambos utilicen el mismo modelo de objetos, dado que en cada uno se puede detectar los roles de Model, View y Controller, las funcionalidades de cada uno son muy diferentes. Para el caso específico del controlador, en el MVC tradicional, éste tenía la responsabilidad de interactuar de múltiples formas con el usuario, mientras que en el MVC Web, su rol es generalmente reducido a tres simples pasos:

1. Leer los parámetros del requerimiento HTTP.
2. Decidir que método del modelo o controlador invocar
3. Decidir que vista es la siguiente a mostrar según el resultado de la ejecución del paso anterior.

Además, en el MVC para aplicaciones GUI se definía que toda la comunicación entre el modelo y las vistas y controladores se haga mediante el mecanismo de observador – observado como notificación de cambios. Ese mecanismo es prácticamente no implementable en la web, y aunque se llegara a efectuar, sería de muy baja performance y con notables demoras.

Por otro lado, el patrón de diseño MVC para la web, es en algunas cuestiones más complejo que el MVC tradicional [14]:

- ***Aplicaciones GUI con modelos pequeños***: Generalmente el modelo de una aplicación GUI es mucho más pequeño que el de una aplicación web.
- ***Reglas complejas para determinar la siguiente vista***: Las aplicaciones web deben aplicar a veces complejas reglas para determinar cual es la siguiente vista a cargar y que datos ella necesita para mostrarse correctamente.
- ***Vista y controlador independientes***: Mientras que en una aplicación GUI, generalmente la vista crea sus controladores, en la web, tanto la vista como el controlador son prácticamente independientes o a lo sumo el controlador es el encargado de crear la vista.

2.4 Ejemplos de MVC en la web

En esta sección introduciré brevemente dos ejemplos de aplicaciones de MVC en la web: JSP Model 2 y Struts. Es necesario aclarar que lo único que se intenta aquí es llegar a presentarle al

lector las mínimas características de ambos frameworks para lograr una implementación correcta de MVC.

2.4.1 JSP Model 2

Como se explicó en el capitulo 1, tanto el uso de Servlet (Sección 1.5: La primera propuesta de Java: Servlets) como JSP (Sección 1.6: La llegada de Java Server Pages) tenía ciertas desventajas que complicaban ciertas tareas durante alguna de las etapas del desarrollo de una aplicación web. Para el caso específico de Servlets, las principales desventajas que se encontraban eran:

- Necesidad de mucho código java para generar la salida
- 100% Código Java que hacía de cambios visuales una tarea no desarrollable por un diseñador de páginas web.

Para remediar esos problemas, Java lanzó JSP logrando con su uso que los desarrolladores, en lugar de encapsular la presentación dentro del código Java, como típicamente se hacía en los servlets, escribir HTML directamente y encapsular el código Java dentro del HTML mediante el uso de tags especiales. Sin embargo, esta tecnología también tenía serios problemas en la presentación: Las páginas JSP tenían demasiado código java generando aplicaciones medianamente difíciles de debaguear y mantener

La arquitectura JSP Model 2 es una combinación de JSP y Servlets que permite el desarrollo de aplicaciones web basadas en MVC. En dicha combinación, JSP brinda el soporte para la presentación de la aplicación mientras que los servlets cubren el rol de controlador, efectuando el procesamiento de la entrada y la llamada a procedimientos del modelo.

¿Como funciona entonces esta combinación "JSP Model 2"? El procesamiento en el servidor comienza cuando el servlet recibe e interpreta el requerimiento de usuario. Una vez interpretado, el servlet se comunica con el modelo mientras llamadas a procedimientos que el modelo publique y luego, a partir de cierta lógica que generalmente se basa en el resultado de la ejecución de los procedimientos del modelo, decide a que vista debe transferir el control. Previo a terminar su ejecución, el servlet recupera del modelo toda la información que la página JSP necesite

mostrar. En este momento es cuando el control le llega a la página JSP. Aquí, la página accede a la información obtenida por el servlet en forma de JavaBean y la muestra.

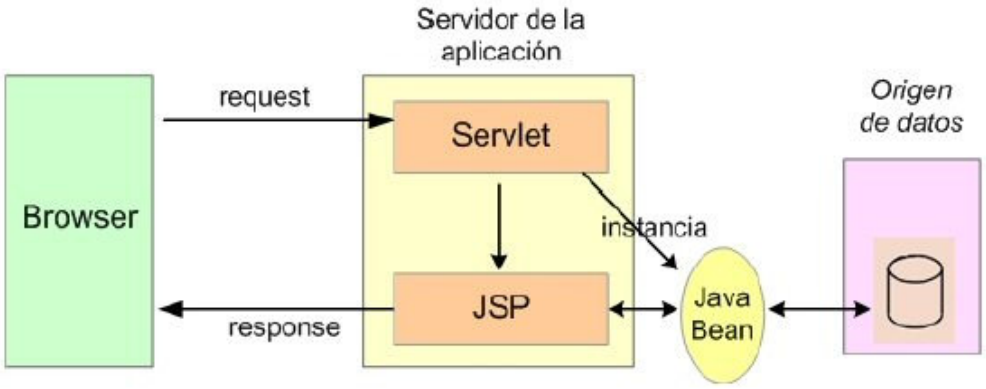


Figura 2.5: JSP Model 2

Como conclusión, JSP Model 2 consigue que la lógica de procesamiento sea eliminada de las páginas JSP, haciendo que éstas tengan únicamente el código java necesario para la recuperación y muestra de la información contenida por los JavaBeans [11] [14].

2.4.2 Struts

Aunque los desarrolladores inteligentes de aquella época notaron que podían combinar Servlets con JSP en lo que luego fue más conocido como JSP Model 2, esa no era todavía la arquitectura óptima para desarrollar aplicaciones web. Si bien las ventajas de cada una de las tecnologías eran utilizadas prácticamente en forma completa en el JSP Model 2, aun así existían ciertos problemas que hacían de éste modelo una implementación de MVC incompleta.

Las principales desventajas del modelo anterior eran:

- Todavía había mucho código JAVA en las páginas JSP lo que hacía que las páginas no sean completamente entendibles por los diseñadores gráficos. Otro problema que llevaba esto era que no había una completa separación entre el modelo y la vista.
- A pesar de que no sea una restricción, sino simplemente una falta de implementación, JSP Model 2 no implementa un `Command` para realizar las llamadas al modelo no consiguiendo tampoco así una separación entre el modelo y el controlador.

2.4.2.1 Definición de Struts

Los problemas enunciados en el párrafo anterior fueron solucionados casi completamente con la aparición de Yakarta Struts. Struts es una framework que implementa Model - View – Controller y cuyo más importante componente es la capa que cumple el rol de controlador definido por el MVC [17]. Esta capa se encuentra desarrollada con una combinación de tecnologías estándares como Java Servlets, JSP, JavaBeans y XML.

Struts no solo provee a los desarrolladores con la función de controlador sino que es un framework que define otros conceptos que ayudan a los desarrolladores y diseñadores a lo largo de todas las etapas del desarrollo de la aplicación. Además del controlador principal, encargado de recibir y dirigir los requerimientos HTTP a objetos manejadores, Struts define ciertas clases que deben extenderse para trabajar y/o extender su funcionalidad

2.4.2.2 Los componentes de Struts

El componente principal del framework de Struts es el `ActionServlet`. Esta clase, que hereda de `Servlet` (del framework de Java Servlets), es la encargada de recibir los requerimientos HTTP y a partir de ellos crear la instancia de acción adecuada que se comunique con el modelo de negocios.

La decisión tomada por `ActionServlet` sobre que acción es la adecuada para atender el requerimiento HTTP se base en un nombre de acción que viene en el propio requerimiento. A partir de dicha nombre y de una colección de instancias de `ActionMapping` accedida por clave se determina cual subclase de `Action` debe instanciarse para efectuar la atención del pedido. Una vez que se conoce la clase, se instancia mediante reflexión y luego se le envía un mensaje para decirle que comience la ejecución encargada de comunicarse con el modelo.

Las subclases de `Action` encapsulan las llamadas a la lógica de negocios del sistema y se encuentran encargadas también de definir cual será la próxima vista a mostrar.

La colección de instancias de `ActionMapping` en la cual se basa la decisión de que acción ejecutar, al igual que otras partes del controlador de Struts como por ejemplo `ActionForward` y

`ActionForm`, son cargadas desde un archivo de configuración basado en XML. Esto permite que el framework de struts sea altamente modificable y configurable logrando que el agregado de nueva funcionalidad no lleve a modificaciones o cambios en los componentes ya existentes del sistema.

Otros componentes del Framework que juegan un papel importante en el proceso de la ejecución y que el usuario necesita conocer son **`ActionForm`** y **`ActionForward`**. **`ActionForm`** es una clase abstracta que el programador debe subclasificar para modelar los objetos que se comunican con la vista, tanto como entrada como de salida. Estos **`ActionForm`** definidos por el desarrollador, deberían estar implementados como JavaBeans, dado que esto permitirá agregarle todas las ventajas que ésta tecnología incluye. Los **`ActionForward`** son objetos que se construyen a partir de un mapeo directo de la sección de Forward definida en el archivo de configuración del framework y que representan las páginas a las cuales se puede transferir el control luego de la ejecución de una acción. Estos **`ActionForward`** permiten no solo abstraer al programador del path de la página pudiendo identificar a cada una a través de un nombre, sino que también consigue que el flujo de navegación del sistema se encuentre definido completamente en un archivo de configuración.

Una funcionalidad interesante que posee el framework de Struts es que, de manera automática y sin la necesidad de que el programador escriba ni una sola línea, instancia y carga las instancias de **`ActionForm`** a partir de los datos del requerimiento HTTP que originó el pedido en el servidor.

En cuanto al componente Vista, las aplicaciones con Struts se construyen generalmente con Java Server Pages aunque el framework soporta otros tipos de tecnologías menos utilizadas. Si bien la posibilidad de generar otro tipo de salida es una gran ventaja, ésta no es la más importante de las ventajas de Struts en el componente visual. Struts provee una importante colección de custom tags que se suman a los existentes en JSP, que permiten eliminar completamente el código Java que en JSP Model 2 se mezclaban con el código HTML. De esta manera, el diseñador puede utilizar la información del modelo sin la necesidad de conocer el lenguaje Java

2.4.2.3 El funcionamiento de Struts

Para entender mejor el comportamiento y la funcionalidad de cada uno de los componentes del framework de Struts, describiré brevemente a continuación los pasos por los que pasa un típico requerimiento web de una aplicación que utiliza Struts [14]. Estos mismos se ven también reflejados en la imagen 2.6, donde al lado de cada paso se puede ver el número de paso que se corresponde en el texto.

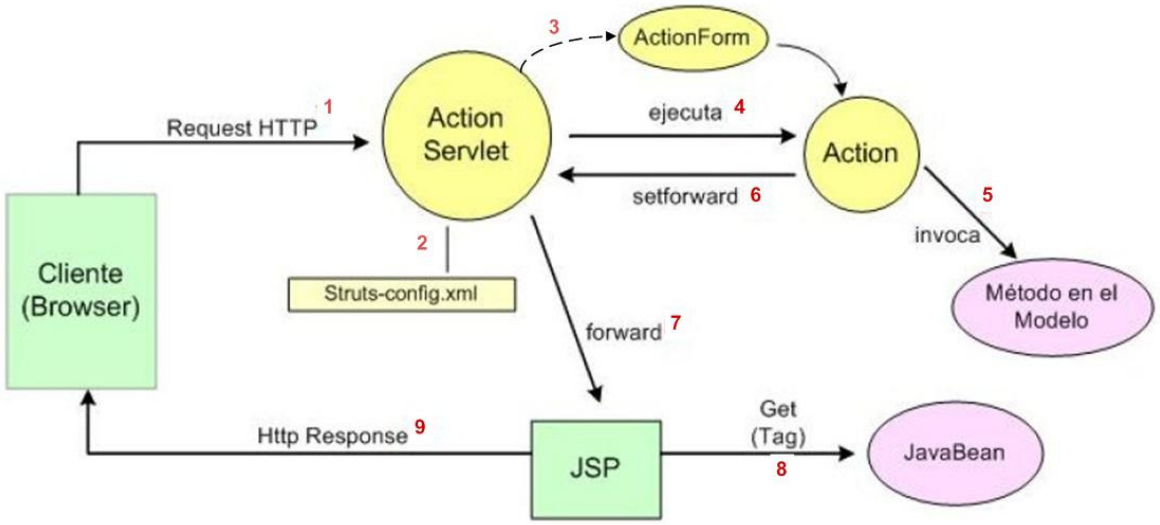


Figura 2.6: Funcionamiento y Componentes de Struts

1. El Servidor web recibe el requerimiento HTTP e inmediatamente es delegado en el **`ActionServlet`**. Una vez que el Servlet tiene el control, procesa el requerimiento y así obtiene la información que viene del cliente web.
2. El ActionServlet obtiene el nombre de la acción que debe invocar desde los parámetros obtenidos en el punto anterior. Con ese nombre de acción y mediante la utilización del archivo de configuración **`struts-config.xml`**, donde se encuentran asociadas las URL con las acciones, se decide que clase de **`Action`** debe crearse.
3. Si en el archivo de configuración se indicase, el **`ActionServlet`** debe crear una instancia de **`ActionForm`** y debe cargar el mismo a partir de la información contenida en el requerimiento HTTP.
4. Una vez que ya se encuentra cargado el formulario, entonces el **`ActionServlet`** le envía el mensaje **`execute()`** a la acción y enviándole como parámetro tanto el **`ActionForm`** como otro tipo de información que la acción pueda llegar a necesitar.

5. Durante la ejecución de la acción, se interactúa con el modelo y se utilizan los datos del **`ActionForm`**.
6. Antes de finalizar la acción, ésta debe seleccionar un **`ActionForward`** que indique hacia que página se debe dirigir la aplicación.
7. Una vez finalizada la ejecución de la acción, el Servlet redirecciona a la página JSP indicada en el **`ActionForward`** seteado en el inciso anterior.
8. La página JSP mediante los tags provistos por Struts interactúa con distintos JavaBeans para completar la información a mostrar.
9. Luego de terminado el cargado de la página, ésta es enviada en formato texto hacia el cliente dentro de una respuesta HTTP la cual es recibida e interpretada por el navegador.

Capítulo 3: ASP.NET

3.1 Presentación de ASP.NET

ASP.NET es más que la siguiente versión de ASP (Active Server Pages): es una plataforma de desarrollo web que provee los servicios necesarios para que los desarrolladores puedan construir aplicaciones web de negocio. Mientras ASP.NET es sintácticamente compatible con ASP, también provee un nuevo modelo de programación y una infraestructura para que las aplicaciones sean más estables, seguras y escalables [20].

Mediante el uso de Web Forms, que mas tarde explicaremos de manera detallada, ASP.NET, permite la construcción de potentes páginas web basadas en formularios. Por otro lado, con la utilización de los controles de servidor (Server Controls), permite la creación y programación de elementos de interfaz comunes para determinadas tareas. Estos controles tienen como consecuencia una construcción rápida de Web Forms a partir de componentes, creados por el usuario o integrados al mismo ASP.NET framework, simplificando el código de la pagina.

ASP.NET adquirió dos nuevas e importantes funcionalidades:

- ***Aplicaciones para dispositivos móviles***: ASP.NET posee controles, componentes y herramientas que ayudan a la rápida construcción de Aplicaciones Web Mobile para múltiples tipos de dispositivos móviles sin la necesidad de escribir código específico para cada dispositivo en particular. Los Mobile Web Forms de ASP.NET están preparados para reconocer los distintos dispositivos y generar la salida adecuada para cada uno de ellos.
- ***XML Web Services***: ASP.NET provee los medios necesarios para acceder a la funcionalidad del servidor de manera remota. Usando Servicios Web, uno puede programáticamente exponer a clientes su negocio de manera que las aplicaciones se comuniquen e interactúen entre ellas. Los servicios web vía XML permiten el intercambio de datos en escenarios cliente – servidor o servidor – servidor mediante el uso de mensajes HTTP y XML.

El framework de páginas web de ASP.NET es un framework de programación que corre en un servidor web para producir y manejar dinámicamente las páginas de ASP.NET (Web Forms) logrando una plataforma de desarrollo que provee los servicios necesarios para desarrollar aplicaciones web de gran tamaño.

El framework de páginas web de ASP.NET crea una abstracción del modelo tradicional de cliente – servidor Web, de manera que se pueda programar una aplicación web usando los métodos y herramientas tradicionales, soportando así la programación orientada a objetos y el desarrollo rápido de aplicaciones que ninguna arquitectura de aplicaciones web actualmente soporta.

Este framework elimina los detalles de implementación de la separación inherente entre el cliente y el servidor de cualquier aplicación web presentando un modelo unificado para responder a los eventos del cliente mediante métodos que corren en el servidor.

3.2 Características generales de ASP.NET

Las características principales de ASP.NET son:

- El modelo de ASP.NET ha cambiado significativamente del de ASP, haciéndolo mas estructurado y orientado a objetos.
- ASP.NET provee un modelo simple que le permite a los desarrolladores escribir parte de la lógica del sistema a nivel Aplicación como por ejemplo eventos que tengan que ver con la sesión, o cada requerimiento en particular. A su vez, ese modelo puede ser fácilmente extendido por el usuario para proveer la funcionalidad que necesite.
- ASP.NET provee facilidades para hacer un fácil uso del estado a nivel sesión de usuario y a nivel aplicación.
- Al ser parte del .NET framework, ASP.NET obtiene las mejoras en cuanto a performance que el .NET framework posee y su máquina virtual (Common Language Runtime) poseen. Todas las aplicaciones ASP.NET son compiladas en lugar de interpretadas, lo cual permite una codificación fuerte y una compilación a código nativo “Just in time”.
- ASP.NET es completamente flexible y factorizable, con lo cual los desarrolladores pueden agregar y remover módulos que les sean relevantes e irrelevantes respectivamente a la aplicación que estén desarrollando.
- La configuración de ASP.NET es almacenada en archivos xml los cuales tienen como gran ventaja que son legibles por el humano. A su vez, cada aplicación puede tener un archivo de configuración distinto adecuando el mismo a los requerimientos particulares de la aplicación a desarrollar.

3.3 El modelo de código Web Form de ASP.NET

Los Web Form están compuestos por dos partes:

- La interfaz de usuario, que contiene HTML estático o controles de ASP.NET, o ambos a la vez y es almacenada en un archivo con extensión .aspx.
- La lógica del Web Form es una clase que hereda de la clase abstracta "Page" del .NET Framework, que interactúa con el formulario. Se encuentra en un archivo separado (aunque eventualmente puede aparecer en el mismo archivo .aspx) llamado comúnmente como página de atrás ("code-behind") y tiene una extensión aspx.vb o aspx.cs dependiendo si fue escrito en Visual Basic o C#.

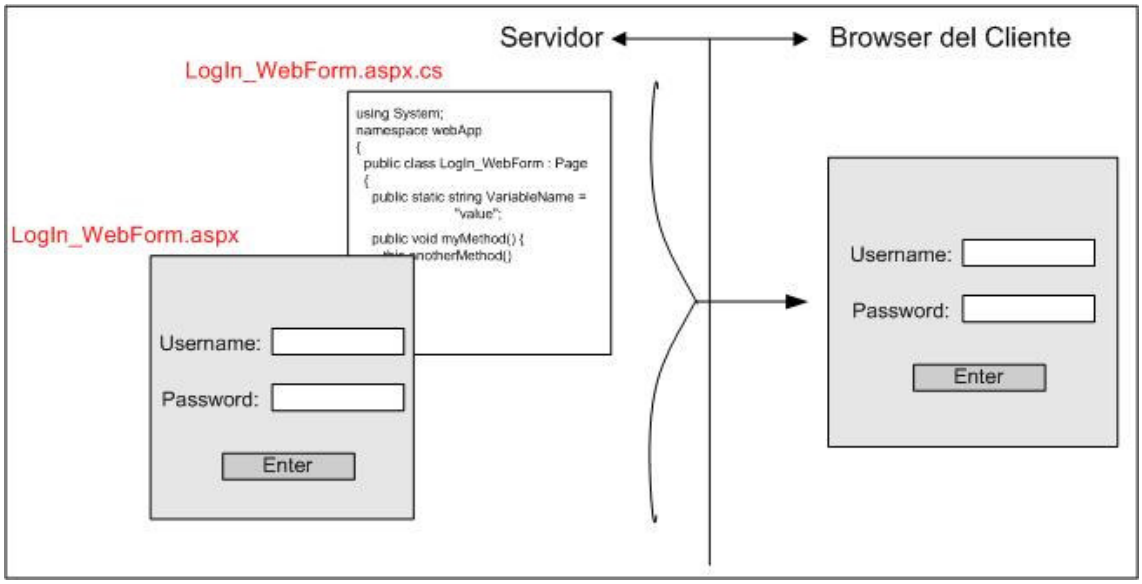


Figura 1: Componentes de un Web Form

En el modelo de páginas ASP.NET, el Web Form entero es en efecto, un programa ejecutable que genera una salida para ser enviada al browser, o dispositivo, que inició el requerimiento. En este modelo, cada página pasa a través de una serie de pasos similar al de cualquier objeto pero con dos pequeñas diferencias:

- La clase Page pasa por esas instancias cada vez que se hace un requerimiento sobre una página. Cada página se inicia, procesa y se libera cada vez que ocurre un roundtrip hacia el servidor.
- La clase Page tiene un paso único, render, que ocurre cerca del final del ciclo de vida de la página, durante el cual se genera la salida que luego será enviada al cliente.

Si bien los Web Form consisten de dos archivos separados, forman una única unidad cuando la aplicación web se ejecuta debido a que todas las clases de las páginas de atrás de la aplicación, son compiladas en una misma dll. Los archivos .aspx son también compilados pero de una manera diferente: la primera vez que el usuario carga la página .aspx, el framework de ASP.NET automáticamente genera un archivo .NET que representa un clase, para luego ser compilada a una segunda dll. La clase recién generada hereda de la clase de la página de atrás que había sido compilada en el proyecto.

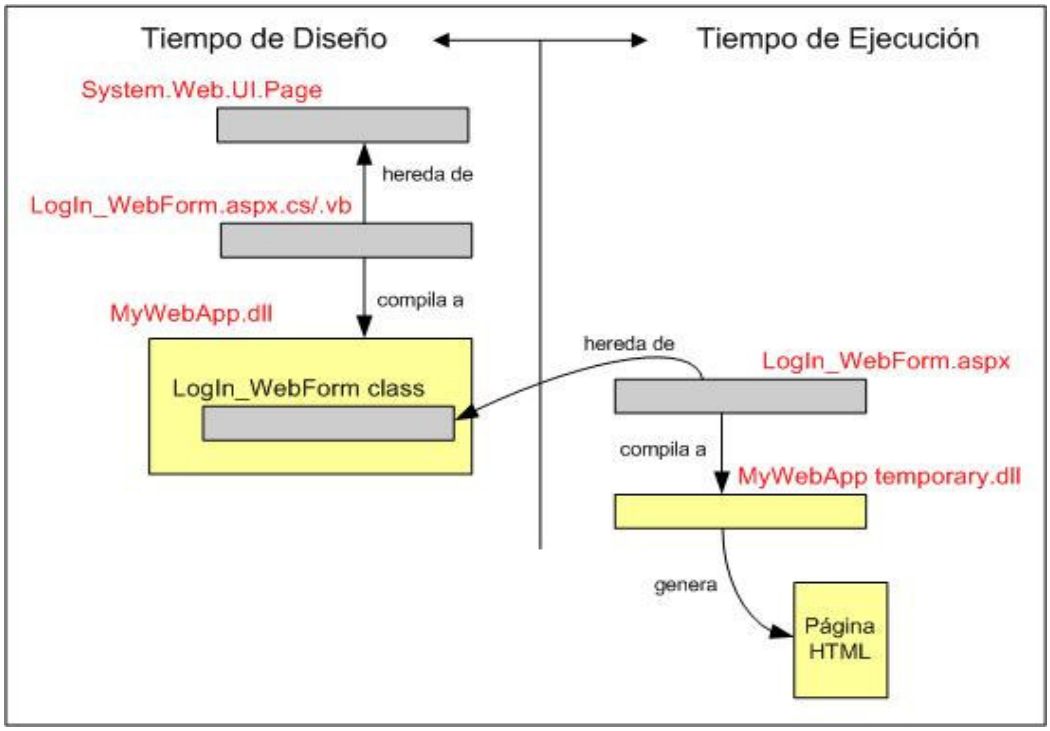


Figura 2: Estructura de un Web Form

Esa dll generada se ejecuta en el servidor cada vez que se peticiona un Web Form, respondiendo mediante la creación de HTML que es enviado nuevamente al browser del cliente.

El modelo de ASP.NET provee un número importante de mejoras en performance y existen dos de ellas que vale la pena mencionar que tienen que se encuentran relacionadas con los requerimientos de HTTP y la compilación de las páginas. Cuando una página ASP.NET es solicitada por primera vez, una instancia de la clase Page es compilada dinámicamente y no interpretada en cada requerimiento como lo hacia ASP. El Common Language Runtime, la máquina virtual del .NET, compila en runtime el código manejado de .NET (bytecodes) a lenguaje nativo del servidor. En segundo lugar, cuando una instancia de página ha sido compilado para el primer requerimiento, es “cacheada” en el servidor, por lo que, para futuros requerimientos de la misma

página, la instancia "cacheada" es ejecutada. Una vez solicitada por primera vez, esa página solo es recompilada cuando el código fuente de la página o uno de sus dependientes (componentes, controles, lógica de negocio, etc.) es modificado [19].

3.4 Web Forms

Los Web Forms son una herramienta que permite crear páginas web programables, realizando la presentación o vista en cualquier browser con HTML y usando código de cualquier lenguaje del Framework (Visual Basic, C#, etc.) para implementar la lógica de la aplicación [18]. Esta separación entre el código y el contenido de la página, elimina el desorden que tenían las páginas ASP.

En general, el ciclo de vida de un Web Form es similar al de cualquier proceso web que corre en un servidor. La naturaleza de no almacenar el estado de las páginas, así como el pedido de información mediante el protocolo HTTP, típicas características del procesamiento web, también se aplican a los Web Forms. Sin embargo, el framework de páginas web de ASP.NET ofrece algunos servicios importantes como transformar o dejar disponible la información de las páginas web en formato de objeto.

El framework de páginas web también mantiene de manera automática el estado de las páginas y la de los controles de las mismas durante el ciclo de procesamiento de la página, es decir, durante distintos pedidos sobre la misma página ("round trip").

Es importante entender el por qué del "round trip" y la división de trabajos en un Web Form. El browser le presenta al usuario un formulario con el cual éste interactúa, haciendo que el formulario haga PostBack al server luego de reaccionar a un evento.

En cualquier aplicación web, las páginas son re-creadas en cada "round trip". Tan pronto como el servidor termina de procesar y enviar la página al browser, descarta la información de esa página. La próxima vez que una página es cargada, el servidor crea y procesa los datos como si fuese la primera vez que se levanta sin tener en cuenta las veces que esta página fue levantada anteriormente, ni el estado que tenía previamente.

En una aplicación web tradicional, la única información que el servidor tiene acerca de un formulario es la que el usuario ha agregado a los controles dentro del formulario, dado que esta es la información que se envía al servidor cuando el formulario es enviado, y cualquier otro dato, como valores de variables y propiedades es desechado.

Estas limitaciones son aprovechadas por el framework de ASP.NET, teniendo como consecuencia un beneficio importante para los desarrolladores de aplicaciones web que usen el framework .NET, de la siguiente manera:

- Almacena la página y las propiedades de los controles entre "round trips", en un lugar llamado comúnmente como ViewState.
- Provee facilidades en el manejo del estado de la página de manera que el usuario pueda salvar sus propias variables entre "round trips".
- Detecta la diferencia entre un Web Form levantado por vez primera de cuando viene de una petición web "PostBack". El usuario podrá programar acciones a ejecutarse al momento de cargarse una página y otras diferentes, si es que lo desea, que se ejecuten al momento de actualizarla.

3.5 Modelo basado en eventos contra modelo lineal de procesamiento

Tanto ASP como por ejemplo PHP utilizan un modelo de procesamiento lineal. Una página de ASP es procesada secuencialmente de arriba hacia abajo donde cada línea de código ASP y/o HTML estático es procesada de la misma manera en que aparece en el archivo. Las acciones de usuario, como puede ser un clic sobre un botón, causan que la página sea enviada hacia el servidor para que este recree la página de salida. Una vez finalizado el procesamiento, la página es recreada y procesada en la misma manera secuencial que antes, mostrando claramente un modelo lineal no basado en eventos.

Para la creación de un modelo basado en eventos, se debe planificar y diseñar bien de manera que se pueda mantener el estado de la página y el control de la misma de una manera fácil de entender y utilizar, sin hacer crecer la complejidad del código.

Un modelo basado en eventos es, sin ir más lejos, el de una aplicación de escritorio en el cual los usuarios interactúan con los elementos, los cuales generan eventos que son manejados por

manejadores de eventos del propio lenguaje. Este tipo de modelo soporta un comportamiento verdaderamente orientado a eventos, en el cual las interfaces gráficas son ampliamente mas ricas reduciendo a su vez la complejidad del código.

ASP.NET reemplaza el modelo de procesamiento lineal de ASP mediante la emulación de un modelo basado en eventos. El framework de páginas de ASP.NET provee implícitamente las asociaciones entre un evento y su manejador, permitiendo que el usuario cree una interfaz que responda a eventos.

3.6 Pasos en el procesamiento de Web Forms

El servidor web carga una página ASP.NET cada vez que se le hace un requerimiento y luego la descarga una vez completado el mismo. La página y sus controles (server controls) son los responsables de ejecutar la solicitud y de hacer el render en HTML para ser enviado al cliente. Con la utilización de ASP.NET y todas sus características, el cliente puede experimentar la sensación de que su aplicación es un proceso continúo de ejecución, olvidándose así de que el servidor no tenga estado y se encuentre además realmente desconectado del servidor. Esta ilusión de continuidad es creada por el framework de ASP.NET y ejecutada por cada página y sus respectivos controles. Cuando ocurre el postback, o rountrip al servidor, los controles se comportan tal como si hubiesen quedado en el mismo lugar donde la anterior solicitud los dejó. Para ello, la propiedad IsPostBack permite saber si la página está siendo procesada por primera vez o viene de un Postback.

ASP.NET procesa las páginas o Web Forms en distintas etapas. Durante cada una de ellas se disparan eventos para que cualquier manejador que se haya registrado como receptor del evento se ejecute. Cada etapa provee al desarrollador, con dichos eventos, de puntos de entradas que permiten actualizar el contenido de una página web.

A continuación se puede ver una lista que contiene las etapas por las que pasa un web form durante su ciclo de vida, una pequeña explicación de lo que cada una de esas etapas representa y los eventos o métodos que se deben atender para poder participar de la etapa.

- **Inicialización** *(Evento Init o método OnInit)*: Inicializa los objetos necesarios para satisfacer lo que suceda durante todo el ciclo de vida del requerimiento web.

- **Carga del ViewState** *(Método LoadViewState)*: Durante esta etapa, la propiedad ViewState es cargada.
- **Carga** *(Evento Load y Método OnLoad)*: Realiza tareas comunes a todos los requerimientos que se puedan hacer sobre una página. En este momento, los controles del servidor se encuentran creados, inicializados y cargados con los datos que poseía el cliente al momento de enviar la solicitud.
- **Manejo de Eventos de Postback** *(Eventos de Server clicks)* Maneja los eventos generados en el cliente que causaron el postback hacia el servidor.
- **PreRender** *(Evento PreRender)*: Realiza cualquier actualización o modificación necesaria antes de generar la salida a enviar al cliente.
- **Render** *(Método Render)*: Genera la salida a ser enviada al cliente.
- **Descarga** *(Método Dispose)*: Realiza las tareas finales como liberación de recursos antes que se le haga dispose a los controles de la página.

El framework de ASP.NET permite de manera relativamente fácil realizar el manejo y control del estado de la página y de sus controles para soportar ese virtual proceso continúo de ejecución. Para ello, los desarrolladores deben entender que información se encuentra disponible para un control durante cada una de las etapas del ciclo de vida de un webform, así como también que datos son persistidos y en que momento, y en que estado quedan los controles al momento de hacer el render de la página [19].

3.7 Modelo de Eventos de ASP.NET

Los eventos generados por los controles y las páginas de ASP.NET son algo diferentes de los eventos generados por una aplicación de escritorio normal y también de los generados por una aplicación web normal. La diferencia ocurre principalmente en la separación de la generación del evento del lugar donde el mismo es manejado.

En las aplicaciones de escritorio o en las aplicaciones web normales, los eventos son generados y manejados en el propio cliente, mientras que en los Web Forms, los eventos asociados a controles del servidor son levantados en el cliente web pero manejados en el servidor web por el framework de ASP.NET.

Para los eventos disparados en el cliente, el modelo de eventos de los controles de ASP.NET necesita que la información del evento se capture en el cliente y sea enviada al servidor mediante un post de HTTP. Una vez en el servidor, el framework interpreta el post determinando que evento ocurrió y así saber que métodos del código del usuario llamar para manejar el evento.

ASP.NET maneja virtualmente todos los mecanismos de capturado, transmisión e interpretado de los eventos. Al momento de la creación de un manejador de un evento en un Web Form, el desarrollador puede enfocar su atención al evento en sí sin preocuparse sobre los mecanismos de cómo el evento fue capturado y puesto a disposición del mismo, tal como lo hace en una aplicación cliente tradicional.

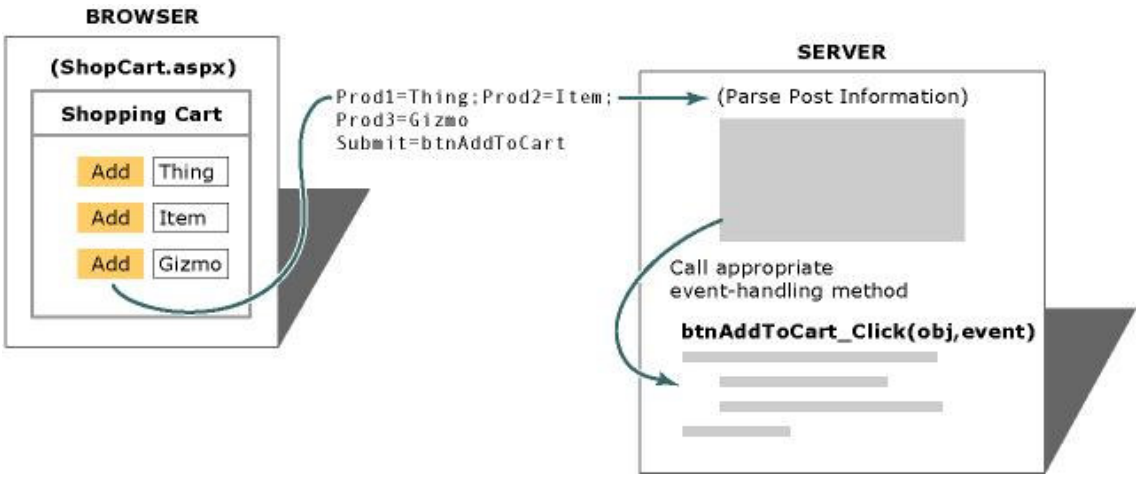


Figura 3: Modelo de Eventos de ASP.NET

Por último, además de los eventos de los Web Forms y sus respectivos controles web, el framework de ASP.NET provee una serie de eventos adicionales que permiten al desarrollador realizar acciones relacionadas con cada requerimiento web, con la sesión de usuario y con la aplicación permitiendo de esta manera:

- Ejecutar código antes de que un determinado requerimiento se comience a procesar, antes o después de ser autenticado, una vez que la instancia de Page ha sido creada pero antes de producirse el load e inmediatamente antes de enviarse la salida al cliente para poner algunos ejemplos.
- Realizar alguna operación cuando el usuario inicia o finaliza una sesión.
- Efectuar acciones cuando la aplicación se inicia o se detiene.

Mediante el uso de cualquiera de estos eventos, el desarrollador puede tener un control más personalizado sobre la aplicación web en general abstrayéndose así de un Web Form en particular, pudiendo aplicar cuestiones de seguridad, control, logueo o iniciación de recursos para toda su aplicación web.

Segunda Parte

Capítulo 4: WMVC Framework

4.1 Introducción al WMVC Framework

El WMVC Framework, como su nombre, lo indica es un framework basado en el patrón de diseño MVC (Model – View – Controller) para el desarrollo de aplicaciones Web. El "Web MVC Framework" (WMVC) provee la separación del modelo de la interfaz mediante la existencia de un controlador y aísla totalmente el lenguaje de la aplicación del lenguaje usado para el diseño de páginas web. El fin de este framework es permitir desarrollar, diseñar y mantener aplicaciones web con el menor costo posible.

Este framework se encuentra dividido en tres capas que separan la aplicación web del modelo subyacente, tal cual lo define el patrón Model - View - Controller. Estas capas son:

- ***Páginas web***: A diferencia del framework de ASP.NET, el WMVC Framework permite escribir páginas web sin la necesidad de conocer ningún lenguaje del Framework .NET. El diseñador de páginas web necesitará solamente conocer el lenguaje HTML, y la biblioteca de Controles (Tags) provista por el Framework.
- ***Controlador***: El controlador del WMVC Framework es su herramienta principal. Mediante el uso de acciones definidas por el desarrollador, el controlador atiende los pedidos provenientes de las páginas web e interactúa con el modelo para satisfacer dichos pedidos.
- ***Modelo***: El Framework permite que el grupo de desarrolladores de la aplicación web se olvide del modelo y de cómo está hecho para centrarse exclusivamente en la funcionalidad que se le debe dar al controlador para que la aplicación funcione.

La subdivisión que propone el WMVC Framework obliga al desarrollador a definir, configurar e implementar un conjunto de acciones en el controlador que conectan la vista de la aplicación con su modelo sin que exista conexión directa entre ellos. El diseñador de páginas web solamente necesita conocer los nombres de las acciones, que no son los de las clases de las acciones sino los que se definieron en los archivos de configuración xml. De esta forma el diseñador web no necesita saber que es lo que realmente hace la acción, ni que pasa una vez ejecutada la misma.

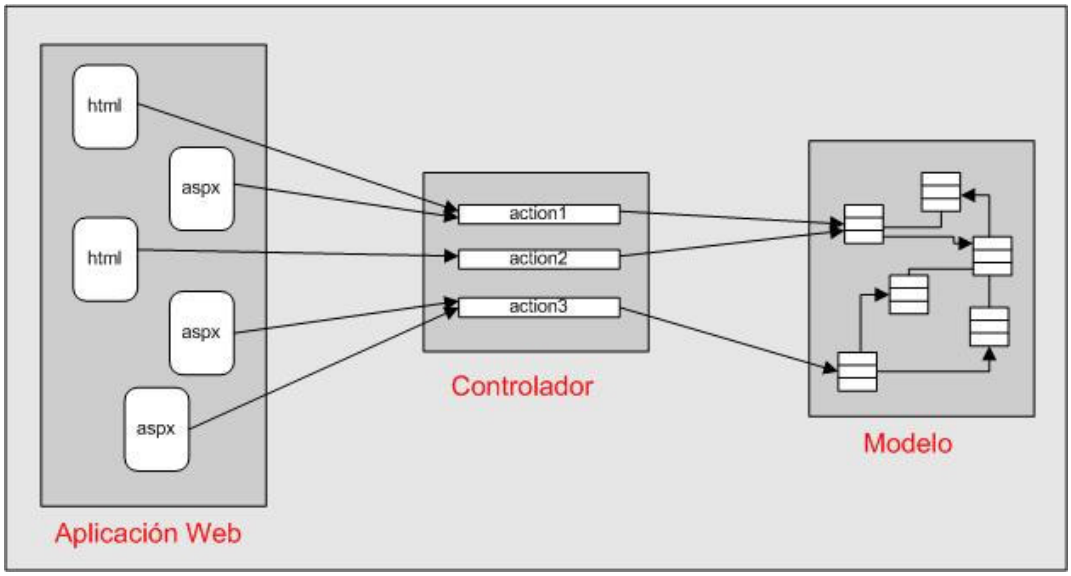


Figura 1: Modelo de una aplicación web usando el WMVC Framework

El WMVC Framework provee además, un conjunto de herramientas que permiten, al equipo de desarrolladores de una aplicación web, una fácil, rápida y ordenada forma de construirla. La herramienta básica del Framework es el "WMVC Framework Core", una dll que posee toda la funcionalidad del framework.

Otra de las herramientas que forman parte del WMVC Framework es la biblioteca de controles web. Allí se encuentran definidos todos los controles web (Web Controls) necesarios para que el diseñador de páginas web construya las páginas web como si estuviese trabajando con los controles del framework .NET.

Por último, existen otro conjunto de herramientas que son externas al funcionamiento del framework en si y que solo complementan al mismo. Estas herramientas externas, tienen como objetivo mejorar el proceso de la construcción del software.

4.2 Componentes del WMVC Framework

El WMVC Framework contiene las clases necesarias para que el framework funcione y un conjunto de herramientas para un desarrollo rápido, fácil y ordenado de la aplicación web en cuestión.

El WMVC Framework se divide en tres componentes generales:

- El Framework Core
- La biblioteca de Controles
- Herramientas adicionales

4.2.1 El Framework Core

El Core del Framework es el corazón o núcleo del WMVC Framework y es la parte del framework que tiene toda la funcionalidad. El Core es un archivo dll (WMVC.Framework.dll) donde se definen todas las clases que le permiten al framework atender los pedidos de ejecución de acciones desde el cliente y levantar la configuración realizada por los desarrolladores que mapea los pedidos del cliente con la acción exacta que debe manejar esa petición.

El Core es quien realmente implementa el patrón de diseño MVC y por lo tanto se encuentra dividido en tres capas:

- Modelo
- Páginas web
- Controlador

4.2.1.1 Modelo

A diferencia quizás de otras aplicaciones web, en nuestro nuevo Framework cualquier modelo se adapta automáticamente a una aplicación web. El WMVC Framework permite que el grupo de desarrolladores de la aplicación web no tenga que conocer el modelo ni cómo fue construido para centrarse exclusivamente en la funcionalidad que le debe dar al controlador, la cual hará que la aplicación funcione. Los desarrolladores solo deben conocer los puntos de entrada al modelo para llamarlos desde el controlador y dedicarse únicamente a construir las acciones del controlador.

4.2.1.2 Páginas Web

A diferencia del framework de ASP.NET, el WMVC Framework permite escribir páginas web sin la necesidad de conocer ningún lenguaje del Framework .NET. Así, el diseñador de páginas

web necesitará conocer solamente HTML y la librería de controles provista por el WMVC Framework.

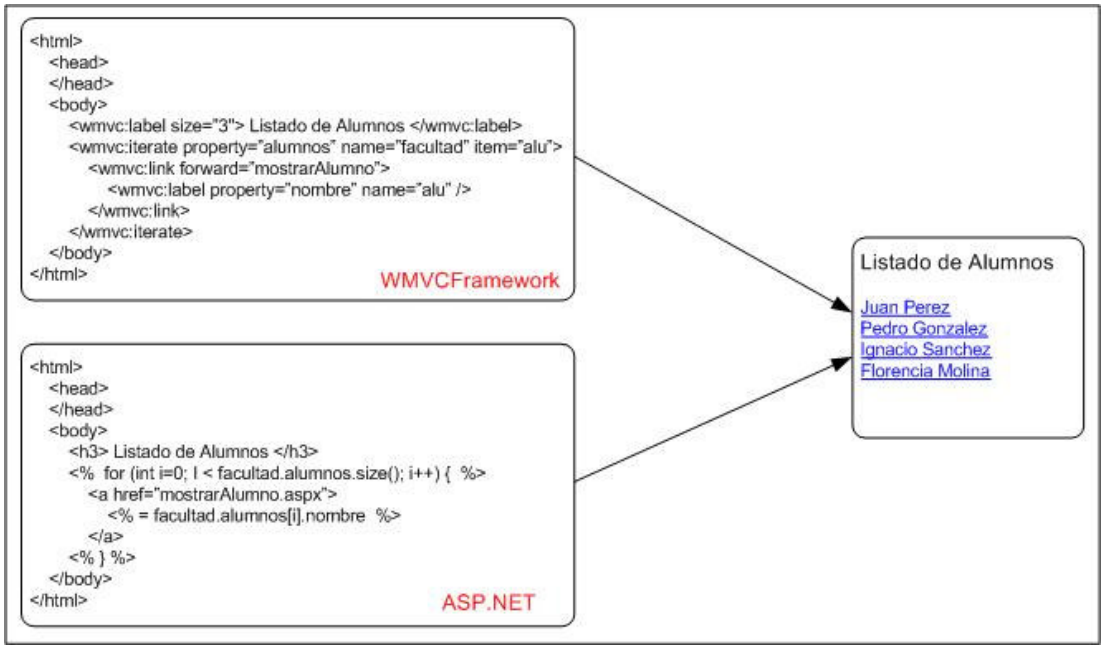


Figura 2: Diferencia entre una página hecha con el WMVC Framework y otra usando ASP.NET

Para mostrar la enorme diferencia que existe al hacer una página usando los controles del framework y otra que no, se muestra, en la figura 2, un ejemplo en el cual se desea mostrar un título que diga “Listado de Alumnos” y luego una lista de links (con el nombre del alumno como texto) a las páginas de cada alumno.

4.2.1.3 Controlador

El controlador del WMVC Framework es la herramienta principal y es la capa que hace la mayor diferencia entre una aplicación basada en ASP.NET y otra que utilice este framework. El controlador atiende los pedidos provenientes de las páginas web e interactúa con el modelo para satisfacer dichos pedidos.

Cada aplicación web que se construya usando el WMVC Framework está dividida en un conjunto de acciones y un conjunto de formularios asociados a las acciones. Además, tanto cada acción como la aplicación web entera tienen definido un conjunto de estados en los cuales puede quedar la aplicación luego de la ejecución de una determinada acción. Esta descomposición del

controlador en varias partes permite al usuario configurar la aplicación a su gusto de manera muy fácil y también tener toda su aplicación modularizada de manera que el mantenimiento de la aplicación también sea sencillo.

En las aplicaciones que usen el WMVC Framework, el flujo de la misma será mediado por un controlador central. El Controlador delega las peticiones a un manejador adecuado, el cual detecta qué acción se está peticionando y obtiene los datos necesarios para ejecutar la acción. Luego, la acción referencia al modelo y éste toma control de la aplicación. Una vez ejecutada la acción del modelo, el manejador vuelve a tomar el control de la aplicación para más tarde delegar en la vista apropiada. La vista apropiada se muestra haciendo un redireccionamiento que se determina a través de una consulta a un conjunto de tablas de mapeo, cargadas desde el conjunto de archivos de configuración xml.

4.2.2 Biblioteca de Controles

El WMVC Framework para lograr una división completa entre la vista y los demás componentes definió una serie de controles web que facilitan la construcción de las páginas web y mobile con ASP.NET.

A simple vista, se han definido dos grupos de estos controles web:

- Controles visuales para la lectura de objetos y propiedades
- Controles para el flujo de la aplicación

Controles visuales para la lectura de objetos y propiedades

Este tipo de controles extienden a los controles definidos por el framework de ASP.NET agregándole a los mismos la posibilidad de especificar un valor dinámico para mostrar. Estos controles definen por ejemplo un campo de texto editable, una etiqueta de solo lectura, un radioButton y un checkbox pero con la diferencia que todos ellos tienen la posibilidad de seteársele un objeto y una propiedad desde donde el valor será obtenido.

Para hacer clara la idea de este tipo de controles supongamos que se desea mostrar en una página de cambio de contraseña, el nombre y apellido del cliente como titulo y luego el nombre de

usuario y la contraseña del mismo en campos de textos editables. El código fuente del archivo .aspx de dicha página se encuentra en la figura 3.

```
<H1> <WMVC:Label object="user" property="FullName" /> </H1>
<P>
  Nombre de Usuario: <WMVC:TextBox id="userUsername" runat="server"
                                   object="user" property="Username" />
  <BR>
  Contraseña: <WMVC:PasswordBox id="userPassword" runat="server"
                                object="user" property="Password" />
</P>
```

Figura 3: Ejemplo de controles web del WMVC Framework

Controles para el flujo de la aplicación

Estos controles son utilizados para evitar codificación de control de flujo tanto sea en la página web (.aspx) como en su página de atrás (aspx.cs o aspx.vb). Con esta forma de codificación de las páginas web se consigue que sea el mismo diseñador el que realice la carga completa de la página sin la necesidad de un programador que genere el código fuente en un lenguaje de programación. Un ejemplo de estos controles se ve en la figura 2 de este capítulo donde en lugar de codificar la iteración sobre la colección de alumnos, simplemente se utiliza el control de iteración provisto por el framework.

4.2.3 Herramientas adicionales

Las herramientas adicionales del framework, tal cual se mencionara antes, tiene como objetivo principal lograr una mejora en el proceso de construcción del framework. La utilización de este tipo de herramientas permite que el programador obtenga rápidamente su estructura de clases y su código fuente base evitando errores en la generación del archivo de configuración, o en la construcción de las clases que extienden al Framework.

Las dos herramientas básicas del framework son:

- ***Configuration tool***: Esta herramienta permite leer y escribir los archivos de configuración a través de una aplicación GUI. La ventaja de esta aplicación es que consigue que el usuario del framework se abstraiga del formato específico del archivo de configuración y se focalice únicamente en generar la estructura de configuración adecuada para su sistema.
- ***Source Code Writer***: Esta aplicación facilita la tarea de los desarrolladores eximiéndolos de la escritura de la estructura de clases que extienden al framework De ésta manera,

una vez definido el archivo de configuración, el desarrollador, mediante el uso de esta herramienta, puede generar todas las clases en forma automática.

4.3 Funcionamiento del framework

Una vez ya presentados todos los componentes del framework y de haber dado una explicación breve de cada uno de ellos nos centraremos en como funciona el Framework y que es lo que el desarrollador debe hacer para utilizarlo. Comenzaré entonces por una descripción de cada uno de los elementos que interesan al funcionamiento para luego hacer una explicación más detallada de cómo es que el framework realmente trabaja.

4.3.1 Descripción de los elementos relacionados con el funcionamiento

El núcleo de funcionamiento del WMVC Framework que cumple el rol de controlador del Model - View – Controller posee ciertos elementos que debemos detallar de manera de que se entienda la existencia de cada uno y que utilidad brinda.

4.3.1.1 Acciones

Una acción en el WMVC Framework, es un conjunto de instrucciones que producen cambios en el modelo de la aplicación, y obtienen los resultados que la vista mostrará en las páginas Web. Estas acciones permiten que la vista de la aplicación no necesite conocer como es el modelo sino solo de qué manera puede interactuar con él.

Estas acciones son implementadas por el usuario y son la implementación del patrón de diseño `Command`. En el caso particular del framework las acciones deben heredar de la clase abstracta `WMVC.Action`. Para llamarlas desde la vista, cada acción posee un texto identificador no dependiente del nombre de la clase de manera de permitir nombres más amigables y además conseguir una separación entre el nombre de las clases y el modo de llamarlas.

Con el objetivo de que sea más claro el concepto de acción, se presenta en la figura 4 un ejemplo de acción para el caso de un logueo de usuario a un sistema.

```
public class LoginAction : WMVC.Action
{
    protected override void execute(ActionRequest request, ActionResponse response)
    {
        LoginObjectModel lom = (LoginObjectModel) request.ObjectModel;
        Model.User user = Model.User.login(lom.Username, lom.Password);
        if (user != null)
        {
            response.ObjectModel = new UserInfoObjectModel(user);
            response.setResultState("userHome");
        }
    }
}
```

Figura 4: Ejemplo de una acción de WMVC

En dicha acción se puede ver como en la primera línea obtiene el object model, para luego en la segunda consultar al modelo de la aplicación. Una vez finalizada la consulta, el modelo retorna un usuario que corresponda con ese usuario y contraseña e inmediatamente es seteado como estado siguiente del sistema **"userHome"** indicando que el usuario se ha logueado correctamente.

4.3.1.2 ObjectModels

Los object models en el WMVC Framework son entidades que solo almacena datos. Estos objetos independizan a los objetos del modelo de la forma en que los mismos son mostrados en la interfaz de la aplicación.

Cada acción puede definir un object model de entrada y uno de salida. El de entrada es cargado en la vista con información que proviene del cliente y es información que la acción necesita para comunicarse con el modelo. Por ejemplo para el caso de una acción de "logueo a un sistema" en la figura 5 se puede ver que el object model de entrada tendría un username y un password ambos en variables de tipo string.

El object model de salida se usa generalmente para pasar información que se desee mostrar desde el modelo hacia la vista. Volviendo nuevamente al ejemplo del logueo al sistema, el object model de salida podría ser una clase que posea información del usuario y un valor booleano indicando el éxito o fracaso de la acción de logueo. En caso que la acción haya finalizado exitosamente entonces información del usuario también le es enviada a la vista de manera que la misma pueda mostrar un mensaje de bienvenida con el nombre completo del usuario.

```
public class LoginObjectModel : WMVC.IObjectModel
{
    protected string username;
    /// <summary>
    /// Get and set user's username
    /// </summary>
    public string Username
    {
        get { return this.username; }
        set { this.username = value; }
    }

    protected string password;
    /// <summary>
    /// Get and set user's password.
    /// </summary>
    public string Password
    {
        get { return this.password; }
        set { this.password = value; }
    }
}
```

Figura 5: Ejemplo de ObjectModel de WMVC

Así, mediante la utilización de estos object models o Data Transfer Object (DTO) se logra cierta independencia entre la vista y el modelo. Las responsabilidades están bien claras: mientras que la acción es la encargada de cargar los object models con la información del modelo, la vista únicamente debe conocerlos y saber mostrarlos.

4.3.1.3 Estados resultantes de la ejecución

Una vez que se concluyó con la ejecución de una acción, ésta debe indicar la siguiente vista a mostrar. Los estados son simplemente ese resultado. El estado representa un nombre que identifique al desarrollador un estado lógico del resultado de la acción que acaba de finalizar. A partir del estado indicado, la configuración obtiene los diferentes tipos de vistas que tiene la aplicación y de esta forma, dado que el requerimiento original sabe a partir de que tipo de vista se generó, se obtiene la siguiente vista.

Para el ejemplo que se viene usando, una vez confirmado el logueo correcto del usuario al sistema, se setea **"userHome"** como estado del sistema, indicando que de la vista de login debe dirigirse a la vista del home del usuario.

4.3.1.4 Vistas

Las vistas son pequeños objetos que tienen como único fin mantener una asociación entre un tipo de vista (web, mobile, etc.) y una dirección que la represente (dirección web para tipos de vista web y mobile, y cualquier otro para otro caso) de manera que, luego de finalizar la ejecución de una acción se determine, según el tipo de vista, cual es la dirección a la que se debe transferir el control.

En el ejemplo del "logueo del usuario", para el caso particular estado resultante `"userHome"` podría indicarse que para una aplicación web se redirija a la página `"home.aspx"` mientras que para una aplicación mobile lo haga a `"mobile/home.aspx"`.

4.3.1.5 Validadores

Los validadores son objetos que en forma parecida a las acciones se ejecutan. La diferencia con las acciones, es que éstos tienen como único objetivo determinar, en forma previa a la acción, si ésta debe ejecutarse o no en el contexto actual.

Ejemplos de validadores podrían ser validaciones de:

- Existencia de usuario logueado en la aplicación.
- Permisos del usuario para la acción a ejecutarse.
- Acciones que se ejecuten solamente para determinado tipo de usuario.
- Verificación de existencia de cierta información en el contexto, como podría ser la necesidad de existencia de información previamente cargada. Un ejemplo podría definirse para la función de modificación de ciertos datos dado que si los mismos no han sido cargados se debe cortar la ejecución de la acción.
- Acciones que se ejecuten solo ante un estado determinado del modelo. Ejemplo de esto podría ser la comprobación de la ejecución de algunos pasos o acciones anteriores en un asistente

Estos validadores permiten que las validaciones no deban repetirse en todas las acciones sino que ocurra antes de las mismas. Así, no solo logramos independizar la validación de la ejecución de la acción sino que además conseguimos que el código de la acción sea más limpio y modificable.

4.3.1.6 Configuración

Dado que el usuario es quien debe definir tanto las acciones, como los object models, los estados y sus vistas, es necesaria la existencia de un método fácil de usar, mantener y aplicar que permita relacionar todos estos elementos.

El WMVC Framework posee un conjunto de archivos en formato xml que permiten, a partir del `wmvc.config` configurar totalmente la aplicación web. Se eligió el formato xml para configurar la aplicación debido a que este formato contiene estructuras legibles y permite que un desarrollador o diseñador de la aplicación pueda configurar la misma sin tener que conocer un lenguaje específico. Por otra parte, la extensión de estos archivos es `.config` debido a que el .NET Framework maneja todas las configuraciones mediante archivos con formato xml pero con la extensión recién mencionada, por lo que era necesario respetar el estilo utilizado por el framework base.

La función del configurador de cada aplicación web es levantar los archivos de configuración e interactuar con el controlador para que éste último ejecute las acciones de manera correcta. Es importante entender la funcionalidad que cumple el configurador dentro del framework ya que todo el funcionamiento de la aplicación depende de ello.

La configuración de una aplicación se encuentra dividida en varios paquetes con el objetivo de separar las distintas funcionalidades de la aplicación web a configurar, de la misma forma que se encuentra dividida en paquetes cualquier aplicación orientada a objetos. Cada paquete representa un archivo de configuración distinto y dentro de cada uno se definen las acciones, object models, estados y subpaquetes del mismo además de la relación que tienen entre ellos y con el modelo de clases (Action, ObjectModel, etc.) que se está configurando.

4.3.2 Interacción entre los componentes

Hemos explicado brevemente en qué consiste cada uno de los elementos que forma parte del funcionamiento del framework, por lo que ahora estamos listos para explicar como interactúan entre ellos y como funciona realmente el framework.

Debido a que los desarrolladores que utilizan el framework definen acciones, object models, estados, vistas y validadores es necesario que exista alguien que organice y coordine el funcionamiento de los mismos. Es aquí donde aparece el Controlador, quien es el objeto encargado de recibir los pedidos desde las páginas web, transmitir al modelo ese requerimiento y finalmente cargar una nueva página web luego de haber ejecutado la acción.

En el dibujo siguiente se muestra un esquema básico del controlador. En la esquina superior izquierda se ve que el requerimiento de la página web llega al controlador. Luego el controlador realiza una serie de pasos y por ultimo se transfiere al navegante a una nueva página web.

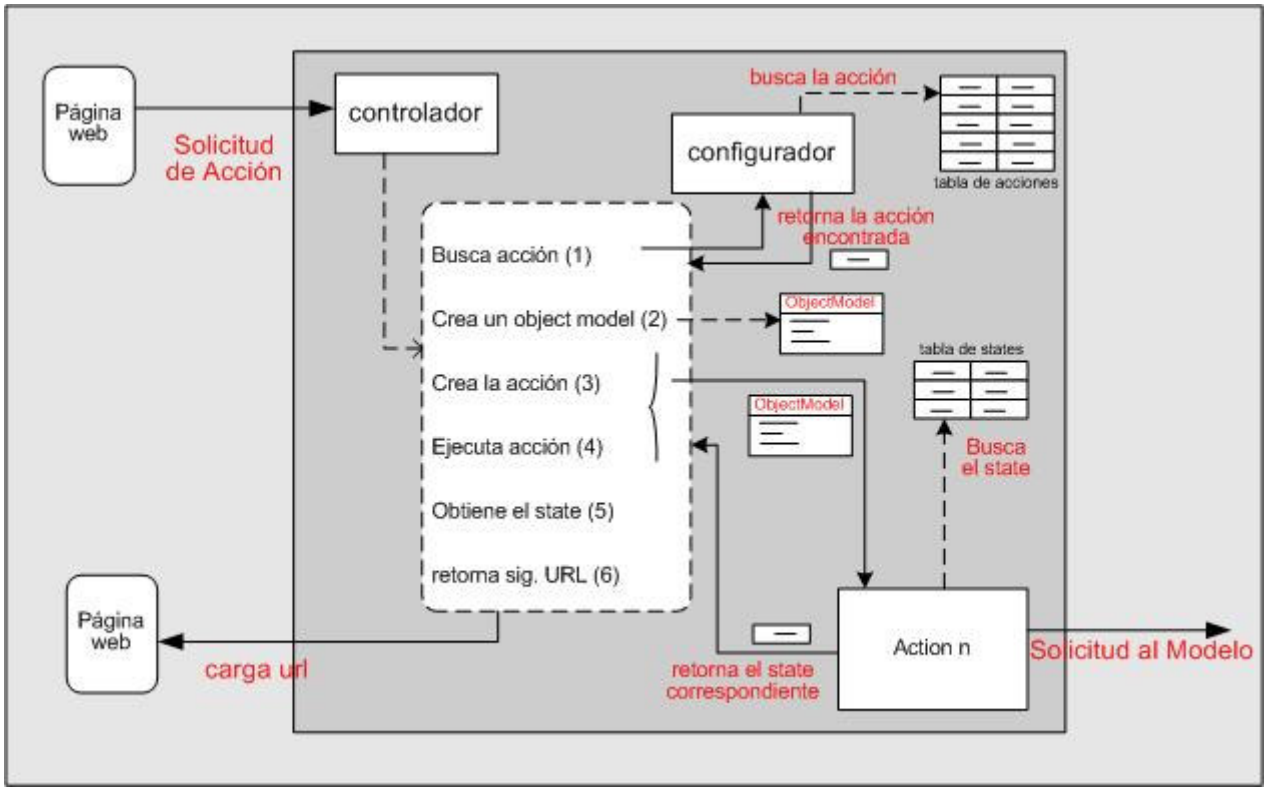


Figura 6: Pasos que efectúa el Controlador en la ejecución de una acción

Los pasos que realiza el controlador para hacer su tarea son los siguientes:

1) ***Buscar acción***: El pedido que realiza la página web al controlador se produce por medio de un texto que representa la acción a ejecutar. Con ese texto, el controlador se comunica con el configurador, el cual le retorna los datos de la acción que debe ejecutar.
2) ***Crear un objectModel***: Luego de obtenidos los datos de la acción a ejecutar, el controlador debe crear u obtener el objectModel necesario para hacer de entrada en la ejecución de la acción.

3) ***Crear la acción***: se instancia la acción a ejecutar y se le pasan los datos necesarios, como el object model y la tabla de estados resultantes en los que puede quedar la aplicación luego de la acción. Estos son necesarios dado que los mismos se utilizan para saber a donde redireccionar según el resultado.
4) ***Ejecutar la acción***: Una vez que la acción ya posee todos los datos que necesita, se ejecuta, lo que incluye una, o varias comunicaciones con el modelo. Antes de terminar, la acción decide el estado en el que finalizó la acción, el cual es retornado hacia el controlador.
5) ***Obtener el estado***: Una vez finalizada la ejecución de la acción, se busca en una tabla de mapeo si el estado seleccionado es válido.
6) ***Retornar siguiente vista***: El tipo de vista que generó la solicitud de acción es utilizado para decidir la siguiente vista que debe ser cargada. Esta decisión es realizada no solo a partir del tipo de vista sino también del estado en que finalizó la acción.

Capítulo 5: Funcionamiento detallado del WMVC Framework

5.1 Instanciación del Framework

El WMVC Framework comienza su funcionamiento apenas se lanza a correr la aplicación basada en dicho Framework. Es en ese momento en el cual el WMVC Framework inicia todos sus componentes y los deja completamente preparados para que cualquier acción desde cualquier tipo de vista lo ejecute.

Durante la etapa de inicialización el Framework realiza la lectura y posterior carga de los archivos de configuración del WMVC Framework. A su vez, inicializa la capa que se ocupará de proveer la separación transparente entre la vista y el modelo

Por otra parte, el componente principal del framework prepara las estructuras y motores necesarios para soportar la ejecución eficiente de cualquier acción que haya sido configurada.

5.2 Eventos a manejar

El WMVC Framework permite durante la ejecución de una acción la intervención, en ciertos momentos, de código del usuario. Para ello, el Framework define tres eventos distintos que el usuario puede escuchar para una o todas las acciones de una determinada página, con el objetivo de permitirle al programador ejecutar ciertas acciones previas y/o posteriores a la ejecución de la acción y previo a la carga de la siguiente vista a mostrar.

Los eventos que define el WMVC Framework son los siguientes:

PreAction: Este evento ocurre justo antes de la ejecución de la acción. Este evento es generalmente usado para cargar los valores del object model desde los controles de la página o los valores que se encuentran en el Request.

PostAction: el evento de PostAction ocurre inmediatamente después de la ejecución de una acción y previo al redireccionamiento de la vista en caso que sea necesaria una transferencia del control desde una vista hacia otra. En este evento, el desarrollador puede, por ejemplo, almacenar manualmente el object model

PostPagePostback: este evento solo sucede a veces. Si la página pierde el control de ejecución debido a que el resultado de la acción obliga un redireccionamiento, entonces este evento no ocurrirá, pasándose directamente al cambio de vista y al evento de carga de la misma que ocurrirá fuera del control del Framework.

Ahora que sabemos que existen esos eventos y en que momento del proceso de la ejecución de una acción se eleva cada uno, pasaremos a ver como un desarrollador debe utilizarlos. Cada clase de vista, sin diferenciar si es mobile, web u otra que haya sido implementada como extensión al Framework, para soportar el WMVC Framework debe implementar una interfaz llamada **`WMVC.IActionCaller`**. Esa interfaz hace referencia a una instancia de **`WMVC.Execution.ActionCallerEventManager`** que es la encargada de interactuar con el desarrollador para determinar los eventos registrados por el mismo.

A partir de esa referencia a la instancia de **`ActionCallerEventManager`** provista por la propiedad **`EventManager`** de la interfaz **`IActionCaller`**, el desarrollador puede seleccionar que eventos necesita ejecutar previa y posteriormente a la ejecución de una acción especifica o para todas las acciones de esa página.

Para la aplicación de ejemplo que hemos venido utilizando durante todo el documento, imaginemos la página donde el usuario visualiza la información de un producto. En dicha página el usuario del sistema podría querer tanto agregar el elemento que esta viendo al carrito de compras como también ver el carrito de compras con sus elementos o visualizar la información de los productos relacionados para ese elemento. Para que la primera acción pueda ejecutarse, necesitamos no solo el producto ha ser agregado sino que también necesitamos al carrito de compras del usuario donde el producto se agregará y al usuario logueado al sistema. Debido a ello, debemos:

- Obtener de la sesión al usuario logueado en el sistema, para que este forme parte del object model a ser enviado hacia el modelo.
- Obtener de la sesión el carrito de compras del usuario donde será agregado el nuevo producto.

¿Cómo hacer esto? Como lo que se necesita debe ejecutarse previo a la ejecución de la acción entonces debemos utilizar el evento de preAction provisto por el Framework para cargar dentro del object model de entrada los valores necesarios.

Supongamos la vista web **viewBook.aspx**; en ella, debemos primero registrarnos al evento preAction de la acción **addProductToCart** para luego escribir el código del evento que obtiene los objetos de la sesión.

```
protected override void registerWMVCEvents()
{
    this.EventManager["addProductToCart"].PreActionExecute +=
        new ActionEventHandler(addProductToCart_PreActionExecute);
}

private void addProductToCart_PreActionExecute(object sender, ActionEventArgs e)
{
    AddToCartOM objModel = (AddToCartOM) e.ObjectModel;
    objModel.User = this.ObjectModelManager.getSessionObjectModel("user");
    objModel.Cart = this.ObjectModelManager.getSessionObjectModel("cart");
    objModel.Product = this.ObjectModelManager.getPageObjectModel("product");
}
```

Figura 1: Ejemplo del uso de los eventos de las acciones

En la figura 1 se puede ver que sobrescribiendo el método **registerWMVCEvents()** provisto por la clase abstracta **WMVC.Web.WebPage** se registran los eventos para las acciones, en este caso particular, el evento **PreActionExecute** para la acción **addProductToCart** mientras que debajo en la misma figura, se encuentra codificado el evento.

¿Cómo implemento los eventos para las acciones? Para conocer la manera de implementar los eventos uno debe comenzar por entender principalmente que es y que contenido tiene el parámetro **ActionEventArgs** de cada evento. Cada instancia de esta clase tiene cuatro propiedades que permiten a partir de este objeto obtener el contexto de ejecución de la acción:

- ***ObjectModel***: Esta propiedad obtiene el object model usado como entrada para el caso del **PreActionExecute** y el utilizado como salida para los eventos de **PostActionExecute** y **PagePostActionExecute**. En caso que en la configuración se haya seteado un object model de entrada o uno de salida entonces este objeto es instanciado u obtenido del lugar especificado automáticamente sin la necesidad de que el desarrollador deba realizar la creación del objeto explícitamente.
- ***ActionMapping***: Esta referencia contiene al **ActionMapping** utilizado por el framework para decidir que clase de acción debía ejecutarse, que object models debían cargarse y si algún validador aplicaba para este caso.

- ***Caller***: Contiene una referencia a la instancia de `IActionCaller` que haya iniciado la ejecución de la acción. Esta propiedad es útil para cuando el manejador del evento se encuentra en otra clase separado de la definición de la vista.
- ***Parameters***: Esta propiedad contiene una colección de parámetros que fueron especificados desde el cliente y que pueden ayudar a la codificación de la acción. Esta propiedad es útil por ejemplo para enviar identificadores, constantes o ciertos elementos que no quieren ser explícitamente agregados en cada ejecución.

Una vez que se entendió como utilizar `ActionEventArgs` la programación de un evento para una acción cualquiera queda tan simple como el código que se ve en la parte inferior de la figura 5.1. En dicho ejemplo, el desarrollador hace un casting a la clase de object model específico de entrada seleccionado en la configuración para luego setearle sus propiedades a partir de valores obtenidos por un `ObjectModelManager` desde distintas partes de la aplicación.

¿Cuál es el rol del `ObjectModelManager`***?*** La responsabilidad que poseen un `IObjectModelManager` es la de interactuar con los medios específicos de almacenamiento de cada tipo de vista y proveer al desarrollador una interfaz casi independiente de la vista actual para la cual se está programando.

Por ejemplo, para el caso de aplicaciones web, el `WMVC.Web.WebObjectModelManager`, interactúa directamente con las clases `Request`, `Page`, `Session` y `Application` provistos por el framework .NET mientras que permite al desarrollador trabajar con un solo objeto al cual le puede pedir, mientras un protocolo extremadamente fácil y claro la obtención o asignación de un object model en un lugar específico. Volviendo al ejemplo de la figura 1 del presente capítulo, en el método `addToProduct…()` se puede ver como obtiene ciertos objetos desde la sesión o la vista actual sin importar exactamente en que tipo de vista se encuentra.

Una de las características que provee esta interfaz es que define un nuevo tipo de tiempo de vida para un objeto: Page. Como se mencionó previamente, el uso de diferentes tipos de `IObjectModelManager` permite olvidarse de las formas de interactuar con las clases específicas dependientes de la vista y permitiendo una mayor dedicación de tiempo a la decisión de donde guardar los objetos.

¿Pero que lugares existen en una aplicación basada en el WMVC Framework donde se puedan almacenar object models o cualquier tipo de objetos? Con la utilización del Framework WMVC para el desarrollo de aplicaciones, el sistema obtiene la posibilidad de almacenar información en cuatro formas distintas:

- ***Application***: Este tipo de alcance permite almacenar objetos que son visibles durante todo el tiempo de vida del sistema.
- ***Session***: Los objetos que son guardados en la sesión tienen como tiempo de vida el tiempo que permanece un usuario conectado al sistema. Una vez finalizada la actividad, ya sea por el propio deslogueo del usuario, por inactividad o por alguna causa imprevista, la sesión finaliza perdiéndose así el pool de objetos almacenados allí.
- ***Page***: Este es el nuevo alcance definido por el framework. Los objetos que sean almacenados con alcance page perdurarán desde que la vista actual es creada hasta que la misma desaparece. Esta "desaparición" no se refiere por ejemplo a una recarga de una página web sino a la transferencia o redirección hacia otra página. [1]
- ***Request***: con la utilización de este alcance los objetos solo permanecen durante un requerimiento, es decir, desde el momento en que la acción comienza hasta que el mismo finaliza.

Una vez conocidas las características principales de esta interfaz y de entender las ventajas que puede tener el utilizarlas, es necesario efectuar una última aclaración. Este `ObjectModelManager` definido como propiedad de la interfaz `IActionCaller` existente en absolutamente todas las vistas de una aplicación es completamente compatible con el resto de la aplicación en las cuales no se utilice el WMVC Framework. Lo que se desea mostrar con esto es que uno podría utilizar, tanto sea para obtener como para almacenar objetos en un object model manager independientemente de si el código que se está ejecutando pertenece a alguno de los pasos que el framework necesita para ejecutarse.

[1] Tipo de alcance Page: Quizás el nombre ideal para este tipo de alcance hubiese sido View dado se refiere a que los objetos almacenados allí permanecen vivos durante toda la existencia de la vista. Como forma de entender mejor este alcance, para las aplicaciones web este alcance permite almacenar objetos y que estos permanezcan vivos o sean recreados en cada viaje al servidor que realiza la página sin necesidad de código extra definido por el desarrollador ni que éste último deba preocuparse por el típico estado desconectado que poseen generalmente las aplicaciones web. Para, por ejemplo, una aplicación WinForm que extienda el WMVC Framework para aplicaciones desktop el alcance View podría definirse como el tiempo que permanece la ventana actual abierta.

¿Qué beneficios tiene el uso de los **`IObjectModelManager`*****?*** Actualmente, que el WMVC Framework no es un producto final para aplicaciones de negocios, sino un framework para facilitar el desarrollo de los distintos tipos de aplicaciones, la utilización de este manager permite:

- Separar la implementación de los eventos a un Assembly distinto al de cada tipo de vista en particular permitiendo de esta manera, una única implementación de todos o por lo menos algunos eventos de las acciones del sistema reduciendo así probables errores y tiempos de desarrollo.
- Abstraerse de la implementación del framework .NET y dedicarse exclusivamente al desarrollo de la aplicación dado que el protocolo provisto es fácil.

Para implementaciones futuras, la utilidad de estos manejadores de object models podría extenderse enormemente dado que, mediante pequeñas modificaciones se podría:

- Hacer que la interfaz de **`WMVC.IObjectModelManager`** sea bastante más compleja permitiendo por ejemplo obtener los valores y propiedades de un control, sin preocuparse por su tipo o por el tipo de interfaz con el que se esté tratando obteniendo así independizarse aun mas de la interfaz.
- Permitir que estos eventos sean asignados mediante la configuración ahorrando de esta forma tiempo de los desarrolladores producto y independizando completamente los eventos de las vistas.

5.3 Distintas configuraciones para a ejecución de una acción

El elemento principal del Framework WMVC es sin duda el concepto de acción. Las acciones son ejecutadas en el servidor por el Framework de ASP.NET y luego derivadas al framework WMVC para que este haga el procesamiento adecuado de las mismas. Sin embargo, la ejecución de las acciones no siempre es igual, sino que depende mucho del contexto de ejecución, el modo en que hayan sido configuradas y la funcionalidad general que se la haya dado a las mismas.

A continuación, se explicarán tres posibilidades diferentes que permiten al desarrollador ejecutar las acciones en diferentes formas:

- ***Modos de acción***: la diferencia entre los diferentes modos permite ejecutar las acciones en diferentes momentos del procesamiento de una pagina ASP.NET.

- ***Nivel de Aislamiento***: los niveles de aislamiento permiten la ejecución de las acciones en diferentes procesos o computadoras.
- ***Jerarquización de las acciones***: mediante la subclasificación de la jerarquía de acciones se puede lograr una funcionalidad general para todas las acciones de una aplicación.

5.3.1 Modo de una acción

El modo de una acción permite básicamente ejecutar una acción en distintos momentos del proceso de ejecución de una página ASP.NET. Actualmente el Framework WMVC provee dos modos de ejecución distintos: OnRequest y OnPostback. Estos modos pueden ser seteados tanto por cada acción como a nivel de paquete permitiendo que, por ejemplo todas las acciones de un determinado paquete se ejecuten en un modo particular. La asignación entre una acción o paquete y modo de ejecución se efectúa estáticamente en los archivos de configuración.

Actualmente, la relación que existe entre una acción y el modo de ejecución es uno a uno pero en versiones futuras se podría llegar a permitir que el modo de ejecución de una acción específica dependa de alguna condición a ser evaluada cada vez que se necesite ejecutar la acción.

5.3.1.1 Modo de acción onRequest

El modo onRequest permite ejecutar una acción antes de que ocurra la carga del mecanismo de Postback provisto por el .NET Framework. La principal ventaja que posee este modo de ejecución es la performance dado que la ejecución de la acción estaría ocurriendo previamente a la creación y carga de los controles web existentes en la página y previo al evento de carga de la página donde es, generalmente donde se realiza la carga de los datos de la página. De esta manera nos estaríamos adelantando a la mayoría del procesamiento de las páginas ASP.NET consiguiendo un alto rendimiento y obteniendo dos ventajas más:

1. Utilizar el evento de PageLoad para, opcionalmente, ubicar visualmente, si es que es necesario, los datos obtenidos durante la ejecución de la acción.
2. Registrarse al evento de `PagePostActionExecute` y de esta manera poder realizar alguna codificación particular post PageLoad que la ejecución de la acción específica necesite.

No obstante, este modo de ejecución tiene ciertas restricciones. La más importante de ellas es que puede ocurrir que para la ejecución de alguna acción necesitemos efectuar la carga de solo algunos componentes de la página y esto no seria posible dado que las acciones que utilizan este modo corren previo a la carga de los controles de la página que generalmente ayudan a la carga de la información de la misma.

También como desventaja nos encontramos con que el desarrollador para obtener la información existente en la página debe utilizar el Request de la misma llevando no solo a incomodidad para programar la obtención de la información sino además en que de esta manera se pierde muchas de las ventajas de la utilización de paginas ASP.NET como son la utilización de los controles.

5.3.1.2 Modo de acción onPostback

El modo de ejecución OnPostback permite ejecutar las acciones en forma posterior a la carga de la página. La ventaja más importante que poseen las acciones que utilizan este modo de ejecución es que pueden utilizar en forma completa todas las ventajas provistas por el framework .NET.

Como principal desventaja frente al modo de ejecución onRequest nos encontramos la performance. Este modo de acción tiene simplemente la misma performance que el framework de ASP.NET mientras que con el modo onRequest se conseguía una mejora substancial en la velocidad de procesamiento de un requerimiento web que sea una solicitud de una acción WMVC.

5.3.2 Nivel de aislamiento

Los niveles de aislamiento son un concepto del WMVC Framework que permite ejecutar las acciones dentro del mismo proceso donde se encuentra corriendo la vista, o en otro proceso, ya sea manejado y controlado por el servidor web Internet Information Server o en un proceso aparte que se encuentre escuchando por un puerto particular.

La ventaja principal de este concepto es que permite separar por completo al modelo de la aplicación de la vista y el controlador. De esta forma el modelo de un sistema se puede encontrar en

un proceso externo dentro del mismo servidor o directamente en otro servidor aparte del que se ocupa de procesar la entrada del usuario y generar la salida correspondiente.

Esta forma de ejecución de acciones, independientemente del tipo de nivel de aislamiento seleccionado es, al igual que con los modos de la acción, configurable en los archivos de configuración y es posible hacerlo tanto por cada acción como a nivel general por paquete y subpaquetes.

Actualmente, el WMVC Framework cuenta con tres tipos diferentes de niveles de aislamiento:

- ***None***: Este tipo de aislamiento es el que toma por defecto el framework si ningún otro nivel es asignado y sirve para especificar que ningún tipo de nivel de aislamiento existe entre la vista y el modelo. Es decir, que la acción se ejecuta en el mismo proceso de máquina que lo hace la vista.
- ***ASP.NET***: Este tipo de aislamiento indica que el modelo se encuentra cargado dentro del Internet Information Server y que es mantenido con vida en un Application Domain del propio servidor web. Mediante esta posibilidad la comunicación entre el modelo y la vista se efectúa con la seguridad ya provista y completamente configurable de un servidor IIS.
- ***Process***: El nivel de aislamiento de proceso permite separar el modelo en un proceso aparte no relacionado con el IIS ni con el proceso original donde se encuentra corriendo la vista. La seguridad que pueda existir en la comunicación depende exclusivamente del como el grupo de desarrolladores configure al módulo de Remoting encargado de efectuar la transmisión.

Tanto para cuando se utilice el nivel de aislamiento de ASP.NET como el de proceso, a partir de la configuración del usuario se podría conseguir, de manera muy simple, separar al modelo no solamente en otro proceso o dominio de aplicación, sino también en otro servidor o computadora.

¿Cómo se logra eso? Los archivos de configuración en el nodo de aislamiento permiten configurar cuatro propiedades distintas para una acción o paquete determinado:

- ***IP***: dirección IP en la cual se encuentra corriendo el modelo.

- ***Path***: Path interno dentro del servidor donde se encuentra el modelo.
- ***Port***: Puerto por el cual se va a transmitir
- ***Protocol***: Protocolo que se va a usar para la transmisión [TCP / HTTP]

Para separar en otro servidor al modelo simplemente se debe configurar la IP de dicho servidor y el puerto y protocolo que se van a usar para la comunicación.

5.3.3 Subclases de action con mayor funcionalidad

Quizás esta posibilidad de configuración de una acción sea la menos configurable y la que menos tiene se encuentra relacionada con la configuración del propio Framework. Esta posibilidad consiste simplemente en utilizar la herencia en los objetos para redefinir y extender ciertas funcionalidades de las acciones.

Mediante la subclasificación de la clase abstracta `WMVC.Action` de la cual toda acción debe heredar podríamos llegar a obtener ciertas funcionalidades genéricas para un grupo o la totalidad de las acciones.

Por ejemplo, mediante la subclasificación recién propuesta podríamos generar clases abstractas que hagas que las acciones sean:

- ***Transaccionales***: Podría llegar a implementarse una clase abstracta de acción donde se redefine el método ejecutar de la acción para iniciar y finalizar una transacción.
- ***Registradora***: Podría codificarse con cierta facilidad una clase que loguee tanto el llamado a la acción como los resultados de la ejecución de las mismas.
- ***Controladoras de concurrencia***: La implementación de este tipo de acción permitiría el control de concurrencia mediante loqueos o mediante otra política sobre la ejecución de las acciones.
- ***Persistentes***: Podrían implementarse acciones con cierta funcionalidad e inteligencia que logre que las acciones provean de forma casi automática la persistencia de los objetos a utilizar en la acción.
- ***Combinaciones varias***: No solo puede desarrollarse herencias simples como estas sino que también se podría llegar a algo infinitamente mas elevado mediante el uso de un modelo con mayor inteligencia que permita acoplar todas las distintas funcionalidades

necesarias en la ejecución de una acción. Esto sería fácil de hacer mediante la implementación de un modelo donde la acción simplemente representa a una colección de objetos a ejecutarse y cada uno de esos objetos tiene una funcionalidad particular.

5.4 Pasos existentes en la ejecución de una acción

Como se mencionó anteriormente, las acciones son el concepto de mayor importancia en el Framework WMVC y ahora, que ya hemos explicado muchas de los elementos que lo componen estamos en condiciones de explicar en forma breve, cuales son todos los pasos que ocurren al momento de la ejecución de una acción.

Las acciones son iniciadas en el cliente, a partir de la generación de un evento, ya sea un click de un botón o sobre una imagen o cualquier otro evento que pueda generar un evento. Este evento, se procesa en forma general, es decir de la misma manera para todas las acciones enviándose al servidor. Una vez en el servidor, éste es leído por el framework de ASP.NET y procesado por el WMVC Framework para finalmente poder ejecutar la acción.

5.4.1 ¿Cómo funciona WMVC Framework en el cliente?

Los controles web de ASP.NET al ser clickeados ejecutan una función javascript, la cual es muy importante para el funcionamiento de ASP.NET llamada `__doPostback()` cuya tarea es la de setear dos campos `hidden` del formulario web definido indicando en ellos que control generó el evento y algún parámetro si fuera necesario, para luego concluir con un envío del propio formulario hacia el servidor.

El Framework WMVC aplicó para los controles de acción el mismo concepto que con los controles de datos: Extender el comportamiento.

¿De que manera se hizo? Pensando de la misma forma que lo hicieron los que desarrollaron el Framework ASP.NET. Volvamos otra vez a la función de javascript `__doPostBack()`, a los valores que setea cada vez que se ejecuta y al para que son utilizados dichos valores. ASP.NET inicia todo su procesamiento en el servidor a partir de la información que tiene en el primer campo

seteado por la función javascript `__doPostBack()` el cual representa el control que generó el submit del formulario en el cliente.

Con esta información, ASP.NET determina si se ha solicitado una nueva página o se ha hecho un submit de un formulario web a través de la función mencionada. En caso que se verifique la realización de un submit desde un cliente, ASP.NET crea la instancia de página, efectúa todo el proceso con normalidad y en la etapa de "procesamiento de Postback" dispara el evento `click` del control web cuyo identificador había sido encontrado en el campo del formulario que fuese seteado en la función `__doPostBack()` en el cliente.

¿Qué relación tiene todo esto con los controles de acción WMVC? Una vez que ya hemos entendido la importancia de la función `__doPostBack()`: ser la encargada de determinar que cosa se va a ejecutar en el servidor, podemos pasar ahora a los controles de acción de WMVC. Estos controles tienen como información particular el nombre de la acción que se debe ejecutar en el servidor y una lista de parámetros propios de la acción.

¿Cómo hago para que estos valores viajen desde el cliente al servidor? De la misma manera que hace ASP.NET con el identificador del control que genera el evento en el cliente. El WMVC Framework define una nueva función de javascript llamada `__doWMVCPostBack()` donde como primer parámetro del mismo se envía el nombre de la acción que se desea ejecutar y como segundo valor los parámetros de la acción. Esta función, asigna estos valores en un campo hidden del formulario web e inmediatamente ejecuta la función de javascript `__doPostBack()` provista por el .NET de modo que el envío del mismo hacia el servidor se realice de la misma manera que si un postback hubiese ocurrido.

5.4.2 Pasos de ejecución de un requerimiento HTTP en ASP.NET

Una vez que el requerimiento http arribó al servidor, éste es procesado. El procesamiento de un requerimiento en ASP.NET involucra una serie de pasos donde cada uno de ellos dispara un evento de manera que el desarrollador pueda captar los mismos en el momento que lo necesiten. En el framework ASP.NET existen dos objetos que escuchan los eventos generados:

- La instancia de página que se está procesando

- Un objeto **HttpApplication** que escucha los eventos independientemente de la página web a procesar.

Los pasos de un requerimiento web en ASP.NET son demasiados como para explicarlos en este documento, sin embargo a continuación describiremos brevemente los que hacen en parte al funcionamiento y/o entendimiento del Framework WMVC:

1. El framework de ASP.NET crea a partir del texto recibido una instancia de **HttpRequest** para procesar el requerimiento.
2. El **HttpApplication** recibe el requerimiento y dispara el evento **BeginRequest**.
3. Se crea la instancia de la página a procesar
4. El **HttpApplication** dispara el evento de **PreRequestHandlerExecute** en el cual se habilita el request en la instancia de página a procesar.
5. La página procesa el evento **OnInit** donde se crean todos los controles web.
6. Se produce el **PageLoad** de la página.
7. Se ejecuta el evento de **Postback** en caso de que la página se esté ejecutando en modo de Postback.
8. La página deshabilita su referencia al **HttpRequest** y al **HttpResponse** correspondiente.
9. La instancia **HttpApplication** encargada de procesar el requerimiento http finaliza su procesamiento y levanta el evento **EndRequest**.
10. La página se renderiza produciéndose los eventos de **PreRender** y **Render** de la página y de sus controles web.
11. El **HttpResponse** resultante es enviado al cliente.

5.4.3 Proceso de ejecución de una acción en el servidor

Una vez explicados los pasos de un requerimiento http para el Framework ASP.NET, ahora pasaremos a ver en forma más detallada los pasos concretos y las tareas por los cuales pasa una acción de WMVC en el servidor web.

El comienzo del proceso de una acción WMVC, ocurre cuando se crea la instancia de página que se va a procesar (Paso 3 de la sección 5.4.2). En este momento:

- Se inicializa un controlador de entrada y salida (**WMVC.IOController**).

- Se instancia un **`WMVC.IObjectModelManager`** que facilitará el manejo de los objectModels.
- Se crea una instancia de **`WMVC.Execution.ActionCallerEventManager`** quien actuará como manejador para los eventos de **`preAction`**, **`postAction`** y **`PagePostAction`** de las acciones.
- Se hace una llamada al método virtual **`registerWMVCEvents()`** el cual permite que el desarrollador en la página concreta efectúe el registro de los eventos correspondientes.

Una vez inicializada la página y registrado los eventos, y correspondiéndose con el paso 4 de la sección anterior, durante el **`PreRequestHandlerExecute`** se determina si el requerimiento es o no un requerimiento de ejecución de una acción WMVC. Si no lo es, se setea en la instancia de la página un flag indicando lo ocurrido pero ante el caso contrario, es decir, que sea un requerimiento WMVC, entonces se obtiene información de la acción a ejecutar y a partir de ello se instancia un ejecutor encargado de ejecutar la acción dependiendo del modo de ejecución de la acción. Toda esa información es almacenada en la instancia de la página de manera que pueda ser accedida en cualquier otro paso del proceso de atención de un requerimiento web.

Una vez que se determinó el modo de ejecución de la acción el procesamiento varía. Si el modo de ejecución es "OnRequest" entonces en el mismo **`PreRequestHandlerExecute`** se dispara el evento de **`PreActionExecute`** haciendo que todos los manejadores que se encontraban escuchando ese evento sean ejecutados.

Luego de finalizados todos los manejadores, se ejecuta la acción, e inmediatamente después se disparará el evento de **`PostActionExecute`** llevando a todos los manejadores de ese evento a ejecutarse.

En caso que la ejecución de la acción haya decidido una redirección hacia otra página, entonces la redirección ocurre en este preciso instante. Aquí es donde se aprecia el porque de la mayor eficiencia de las ejecuciones de acciones en modo **`OnRequest`**:

- Los controles de ASP.NET no han sido creados.
- El **`PageLoad`** ocurre únicamente en caso que se decida no redireccionar.

Durante el **OnInit** de la página se crean todos los controles web y se recuperan los valores de los mismos desde el **HttpRequest** y luego ocurre el **PageLoad**.

Una vez finalizado el **PageLoad**, es el procesamiento del postback quien se hace cargo de tomar el control de todo el requerimiento. Nuevamente esta ejecución depende del modo de ejecución. En caso que el modo sea **OnPostBack**, es aquí donde ocurrirá del mismo modo que ocurriera con el modo **OnRequest** durante el **PreRequestHandlerExecute**, la ejecución en orden del evento **PreActionExecute**, ejecución de la acción, evento **PostActionExecute** y la posible redirección hacia otra página o vista según sea el caso.

Luego esa cadena de ejecución, y solamente cuando no se haya redireccionado hacia ninguna otra vista, será el turno de que el evento de **PagePostActionExecute** sea lanzado. Los manejadores registrados a este evento generalmente se ocuparían de realizar tareas posteriores a la ejecución de una acción como puede ser el recargado de una determinada grilla, la simple verificación de cierta información resultante de la acción, o la carga de sus object models en los almacenes de objetos existentes para posteriores consultas.

Después de que el evento de **PagePostActionExecute** ha finalizado, el proceso de ejecución de WMVC finaliza satisfactoriamente, dejando al Framework de ASP.NET terminar de manera normal el procesamiento del requerimiento http.

Capítulo 6: Framework Core

6.1 División del core en componentes

El módulo principal del framework WMVC es sin duda alguna el Framework Core. Este módulo contiene la funcionalidad más importante del Framework que es la de actuar como mediador entre las diferentes vistas y el modelo del sistema.

El Framework Core es en una vista general de las partes del sistema, el módulo encargado de recibir los requerimientos web, determinar si corresponden con una solicitud WMVC y a partir de ello, procesar los mismos con el objetivo de ejecutar una acción.

En la figura 1 se puede visualizar una descripción gráfica de los diferentes componentes existentes en el Framework Core:

- ***Configuration***: Este componente se ocupa de leer y escribir la configuración del framework como también realiza la interpretación de un pedido de una acción obteniendo el resultado desde la configuración
- ***Transport***: El componente de transporte se encarga de, en caso que sea necesario, proveer el aislamiento necesario entre la ejecución propia de la acción y todos los demás pasos del procesamiento de la misma.

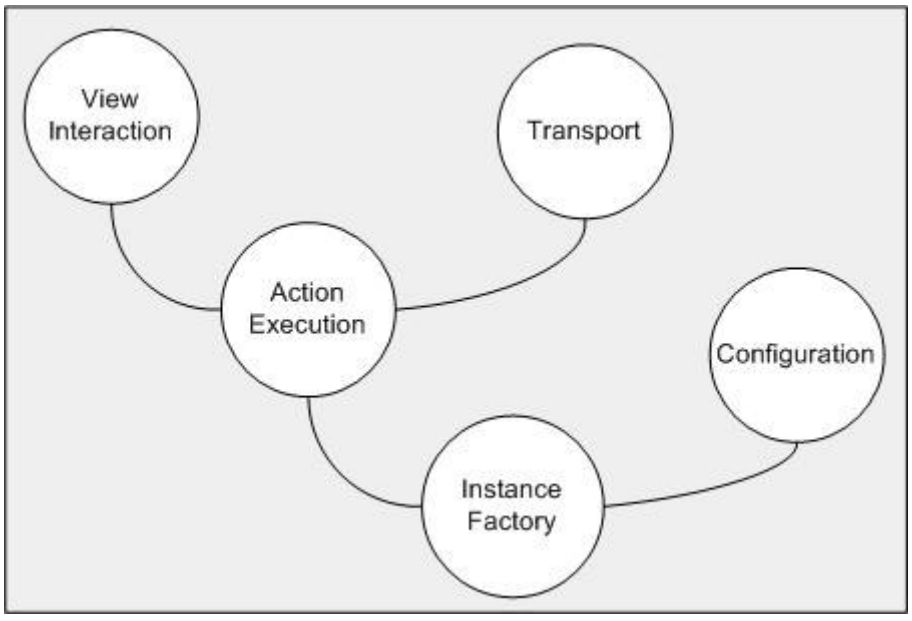


Figura 1: Componentes del WMVC Framework Core

- ***Instance Factory***: el Instance Factory es el componente que se encuentra encargado de la creación de cualquier tipo de objetos. Para ello utiliza tanto información de la acción a ejecutar como la configuración.
- ***View Interaction***: Este componente se encuentra encargado de interactuar con los diferentes tipos de vista con el objetivo de obtener y proveer de información tanto como para que la vista se muestre como para que las acciones del modelo se ejecuten correctamente.
- ***Action Execution***: Esta última unidad tiene como función la ejecución real de la acción.

6.2 Configuración

Por ahora se ha explicado como se configura la aplicación y como funciona el WMVC Framework pero poco se ha explicado acerca de la relación entre los archivos XML de la configuración con el mecanismo que utiliza el módulo de configuración (figura 2) del framework Core para ejecutar la acción solicitada.

En esta sección explicaremos primero, los pasos que ocurren con respecto a la configuración desde el momento de levantar la aplicación y por lo tanto el framework hasta los pasos que ocurren con respecto a la lectura de los archivos de configuración y sus respectivos objetos maleadores al momento en que la primera solicitud de ejecución de acción es efectuada. Luego, pasaremos a detallar la forma en la cual el módulo de configuración interactúa con el resto del Framework, o más específicamente con el `ApplicationController`.

6.2.1 Lectura de los archivos de configuración

Apenas se carga la aplicación en el directorio virtual del servidor de Internet Information Server un evento de la aplicación, el `OnApplicationStart`, es disparado. En ese momento, se instancia un objeto `WMVC.ApplicationController` quien controlará el Framework. Dicho objeto posee una referencia a un `WMVC.Configuration.ConfigurationManager`, el cual es creado automáticamente en el mismo proceso de creación del objeto anteriormente mencionado. La creación de este objeto no involucra más que preparar las estructuras adecuadas para que, apenas se necesite, se carguen los correspondientes archivos de configuración.

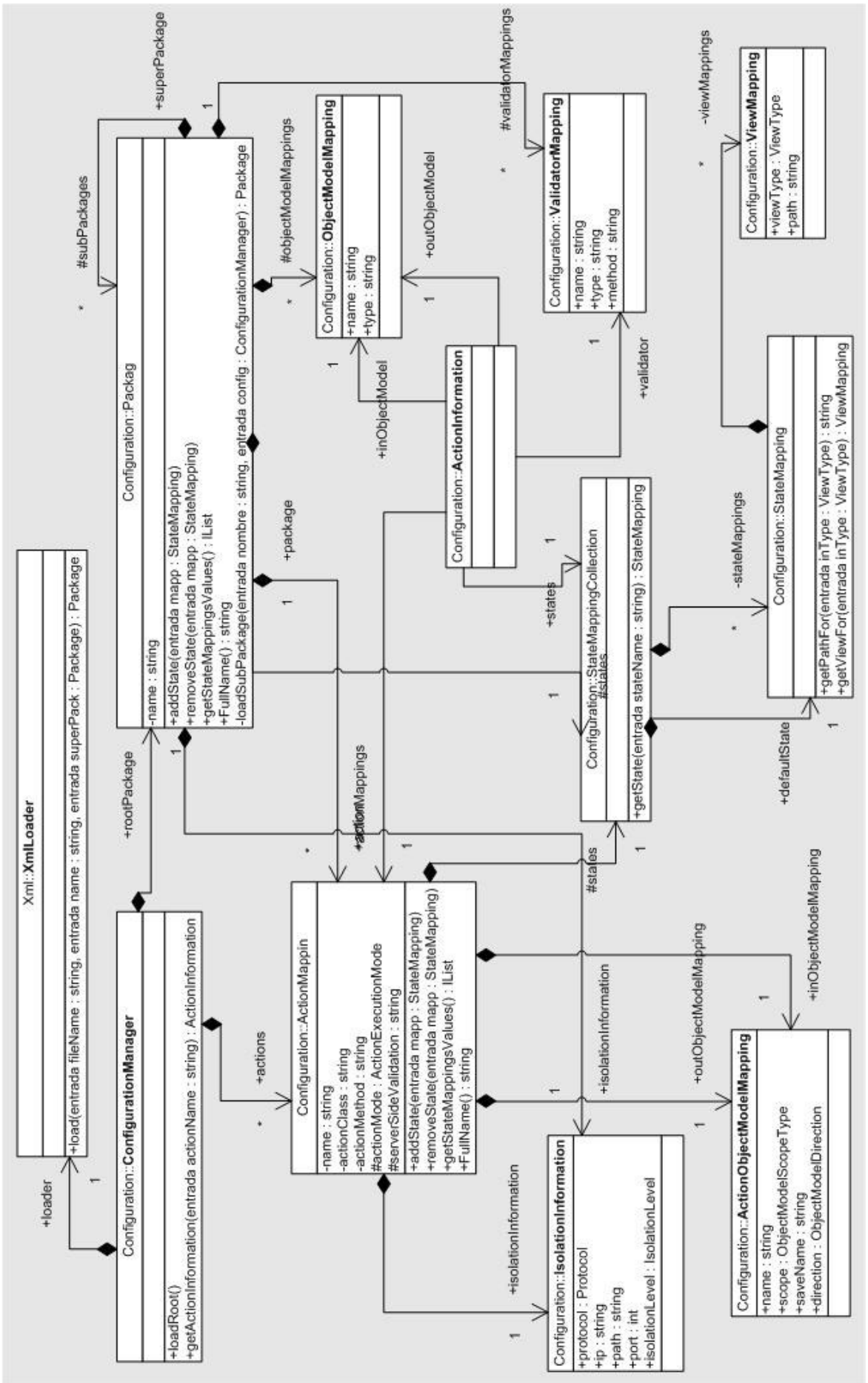
Ante la primera llamada a una acción en la aplicación, el `ApplicationController` le solicitará a su configuración que le provea de información acerca de la acción que debe ejecutar. En este momento, el `WMVC.Configuration.ConfigurationManager` notará que su root package no se encuentra en memoria, por lo que deberá cargar el archivo `WMVC.config` para comenzar a levantar la configuración de la aplicación.

Elemento	*Clase*	*Agregado a:*
Acción	`ActionMapping`	Colección de acciones del paquete.
Object Model de entrada o salida de una acción	`ActionObjectModelMapping`	Referencia al object model de entrada o salida en un `ActionMapping` específico.
Object Model	`ObjectModelMapping`	Colección de object models del paquete.
Estado	`StateMapping`	Colección de estados tanto sea de un paquete como de una acción del paquete.
Vista	`ViewMapping`	Colección de vistas de un determinado estado.
Validador	`ValidatorMapping`	Los validadores de un paquete

Tabla 1: Relación entre los archivos de configuración y el modelo de la configuración

La lectura del paquete principal involucra:

- La creación de una instancia de `WMVC.Configuration.Package`, el cual representa al propio archivo de configuración `wmvc.config`.
- Por cada acción, object model, estado, validador o vista que se encuentre en el archivo de configuración, estos son mapeados a instancias de clases del modelo y luego son agregados a la colección de elementos que corresponda (ver Tabla 1).
- Por cada dll que se encuentre se le indica a su controlador de aplicación que se ha encontrado una definición de un assembly.
- Por cada subpaquete que se encuentre se repite este proceso de carga pero con el paquete referenciado en este subpaquete en lugar de hacerlo con el paquete raíz.


Xml::XmlLoader
+load(entrada fileName : string, entrada name : string, entrada superPack : Package) : Package
+loader
Configuration::ConfigurationManager
+loadRoot()
+getActionInformation(entrada actionName : string) : ActionInformation
+rootPackage
+superPackage
Configuration::Packag
-name : string
+addState(entrada mapp : StateMapping)
+removeState(entrada mapp : StateMapping)
+getStateMappingsValues() : IList
+FullName() : string
-loadSubPackage(entrada nombre : string, entrada config : ConfigurationManager) : Package
#subPackages
#objectModelMappings
#validatorMappings
-viewMappings
+package
Configuration::ObjectModelMapping
+name : string
+type : string
+outObjectModel
+inObjectModel
Configuration::ValidatorMapping
+name : string
+type : string
+method : string
+validator
Configuration::ViewMapping
+viewType : ViewType
+path : string
Configuration::ActionInformation
+actions
+actionMappings
Configuration::ActionMappin
-name : string
-actionClass : string
-actionMethod : string
#actionMode : ActionExecutionMode
#serverSideValidation : string
+addState(entrada mapp : StateMapping)
+removeState(entrada mapp : StateMapping)
+getStateMappingsValues() : IList
+FullName() : string
+states
Configuration::StateMappingCollection
#states
+getState(entrada stateName : string) : StateMapping
-stateMappings
Configuration::StateMapping
+getPathFor(entrada inType : ViewType) : string
+getViewFor(entrada inType : ViewType) : ViewMapping
+defaultState
+isolationInformation
Configuration::IsolationInformation
+protocol : Protocol
+ip : string
+path : string
+port : int
+isolationLevel : IsolationLevel
+outObjectModelMapping
+inObjectModelMapping
Configuration::ActionObjectModelMapping
+name : string
+scope : ObjectModelScopeType
+saveName : string
+direction : ObjectModelDirection

6.2.2 Pedido de información a partir del nombre de una acción

Una vez presentado la información mínima sobre la configuración del Framework, pasaremos ahora a ver el funcionamiento interno del mismo en su componente de configuración.

Como se mencionó en forma breve anteriormente, los archivos de configuración son cargados en clases, cuya mayoría llevan el nombre de **`Mapping`** y que representa en objetos la información textual de la configuración. Estos objetos son creados cuando la primera acción es solicitada y la carga se realiza actualmente en forma completa, logrando de esta manera, solamente demorar el procesamiento de la primera acción. Cuando llegue el turno de ejecutar el resto de las acciones, esto no le consumirá tiempo de carga al módulo de configuración, dado que las configuraciones ya debieron ser mapeadas a objetos previamente.

Una vez que la configuración se encuentra cargada, ésta es manejada, controlada y accedida por una instancia de la clase **`WMVC.Configuration.ConfigurationManager`** la cual se encuentra referenciada desde el **`ApplicationController`** de la aplicación.

Este **`ConfigurationManager`** da soporte al **`applicationController`** mediante objetos **`WMVC.Configuration.ActionInformation`**. Cuando el **`applicationController`** necesita ejecutar una acción que todavía no tiene cargada, éste le solicita a su instancia de **`ConfigurationManager`** que le provea de información sobre esa acción mediante el llamado al método público que se puede ver en la figura número 2.

El manejador de la configuración, a partir de la solicitud por información sobre la acción retorna con una instancia de **`WMVC.Configuration.ActionInformation`** todos los datos que el **`ApplicationController`** necesita para ejecutar dicha acción. **`ActionInformation`** es una clase que representa al nodo Action de la configuración pero modelado completamente en objetos. Así es como contiene, en lugar de los nombres de los componentes que necesita (ObjectModel de entrada y salida, Validator, etc.), directamente referencia a las instancias de **`ObjectModelMapping`**, **`ValidatorMapping`** y la colección de **`StateMapping`**, no solo la que se define en el nodo de la acción sino también los que hereda del paquete y de los superpaquetes.

```
/// <summary>
/// Gets the ActionInformation with all the mapping information for the
/// requested action name.
/// </summary>
/// <param name="actionName">name of the requested acion</param>
/// <returns>Returns the AcionInformation instance</returns>
public ActionInformation getActionInformation(string actionName)
```

Figura 3: Método público del ConfigurationManager

De ésta forma, el controlador de la aplicación únicamente se comunica con su configuración mediante el mensaje **`ActionInformation getActionInformation(string actionName)`** que le devuelve la instancia de **`ActionInformation`** con toda la información que la aplicación necesita para ejecutar la acción. Al conseguir que esta sea la única forma de comunicación entre el componente de configuración y el componente principal del Framework se logra una casi total independencia entre ambos permitiendo que cualquiera de ellos cambie su forma de trabajar sin que la otra parte necesariamente lo note.

6.3 Transporte entre capas

El componente de transporte del Framework Core es el módulo que se encarga de proveer, disponer y asegurar los distintos tipos de aislamiento con que se puede ejecutar una acción. Este módulo actúa a partir de las configuraciones de aislamiento de cada acción y comienza a funcionar desde el mismo momento en que el Framework se inicia.

El módulo contiene a un elemento que tiene como funcionalidad manejar y controlar los diferentes tipos de aislamiento que haya en el sistema. Este elemento, el cual es instancia de **`WMVC.Isolation.IsolationManager`** contiene una colección con objetos que pertenecen a alguna subclase de **`WMVC.Isolation.AbstractIsolationLevel`** las cuales implementan cada una un distinto nivel de aislamiento.

Como en la mayoría de los mecanismos de transmisión que permiten comunicaciones remotas mediante objetos, la comunicación real en tiempo de ejecución es transparente tanto hacia el desarrollador como hacia el usuario y lo único que se necesita para usar el mecanismo es de algún seteo previo entonces se decidió que el WMVC Framework utilizara algo similar.

Cada vez que una acción es requerida por primera vez, es decir, cuando se necesita ejecutar un tipo especifico de acción y no cuando se inicia el Framework, la instancia de

ApplicationController que se encuentra en el repositorio **Application** del servidor web, lo detecta. Entonces, previo a la creación de instancias de esa clase requerida, le informa al manejador de aislamiento se ha detectado un nuevo tipo de acción y que esta debe ser registrada. Es allí cuando este manejador de niveles de aislamiento se comunica con la instancia de una subclase de la jerarquía de **AbstractIsolationLevel**, determinada a partir del nivel de aislamiento especificado por la configuración para esa acción, indicándole que debe registrar al nuevo tipo.

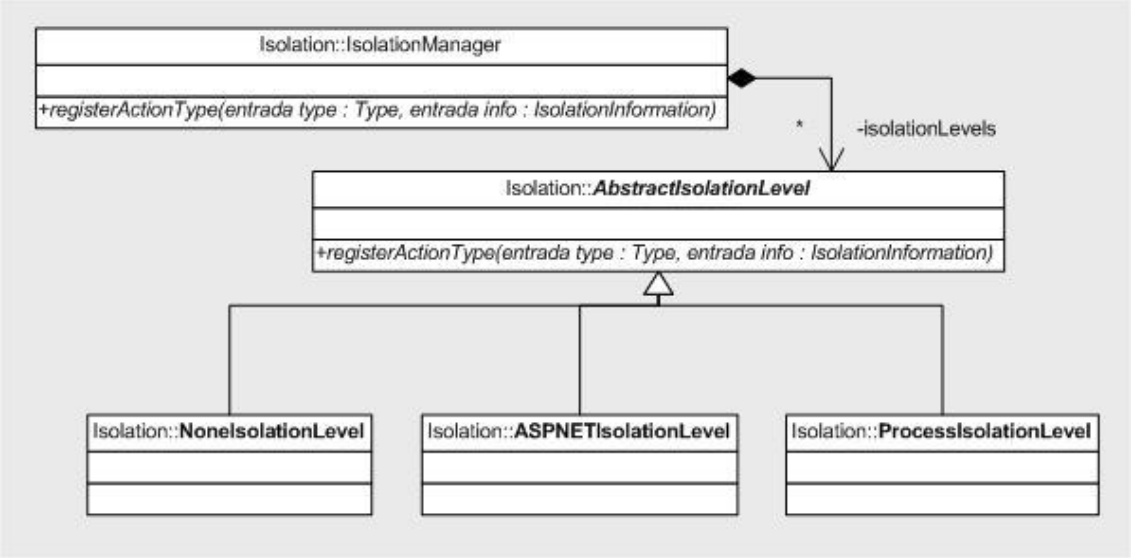


Figura 4: Componente de Aislamiento

Actualmente, el WMVC Framework incluye el soporte para tres tipos diferentes de niveles de aislamiento: None, ASP.NET y Process (explicados en capítulos anteriores), sin embargo, la utilización del patrón de diseño Strategy en la jerarquía provista por este módulo de "transmisión entre capas" permite que cualquier subclase de **AbstractIsolationLevel** pueda ser la encargada de aislar de una manera u otra a las acciones del WMVC Framework. Mediante la simple implementación de una clase que implemente el método:

```
void registerActionType(Type, IsolationInformation)
```

y que herede de **AbstractIsolationLevel** estaría ampliando las posibilidades de aislamiento del Framework.

6.4 Creación de objetos dinámicos

El módulo de Instance Factory o fábrica de instancias tiene como principal responsabilidad la creación de instancias de Acciones, Validadores, Object Models o cualquier tipo de objeto que

deba cargarse en forma dinámica. Para ello, y dado la información que le llega como entrada desde el módulo de ejecución de acciones, debe, frecuentemente comunicarse con el componente de configuración para solicitarle cierta información aun no utilizada por el sistema.

El elemento principal de este módulo es el **`WMVC.Factory.InstanceManager`** y existe una instancia de esta clase por aplicación que utilice el WMVC Framework. Dicha instancia recibe desde el módulo de ejecución de acciones un pedido para obtener un ayudante para ejecutar la acción requerida mediante el mensaje:

```
ActionExecutionInstanceManager getActionManager(string actionName)
```

Este método busca en la tabla de **`actionInstanceManagers`** que posee el propio **`InstanceManager`** la instancia de **`ActionExecutionInstanceManager`** que corresponda con ese nombre acción. En caso de encontrarse, entonces simplemente la retorna dejando que posteriormente sea el modulo de ejecución el que le solicite los elementos necesarios al objeto resultante. Ahora, para el caso en que no se encuentre la instancia buscada entonces, el **`InstanceManager`** deberá solicitarle a su configuración que le provea del **`ActionInformation`** necesario para crear el **`ActionExecutionInstanceManager`** de esa acción específica.

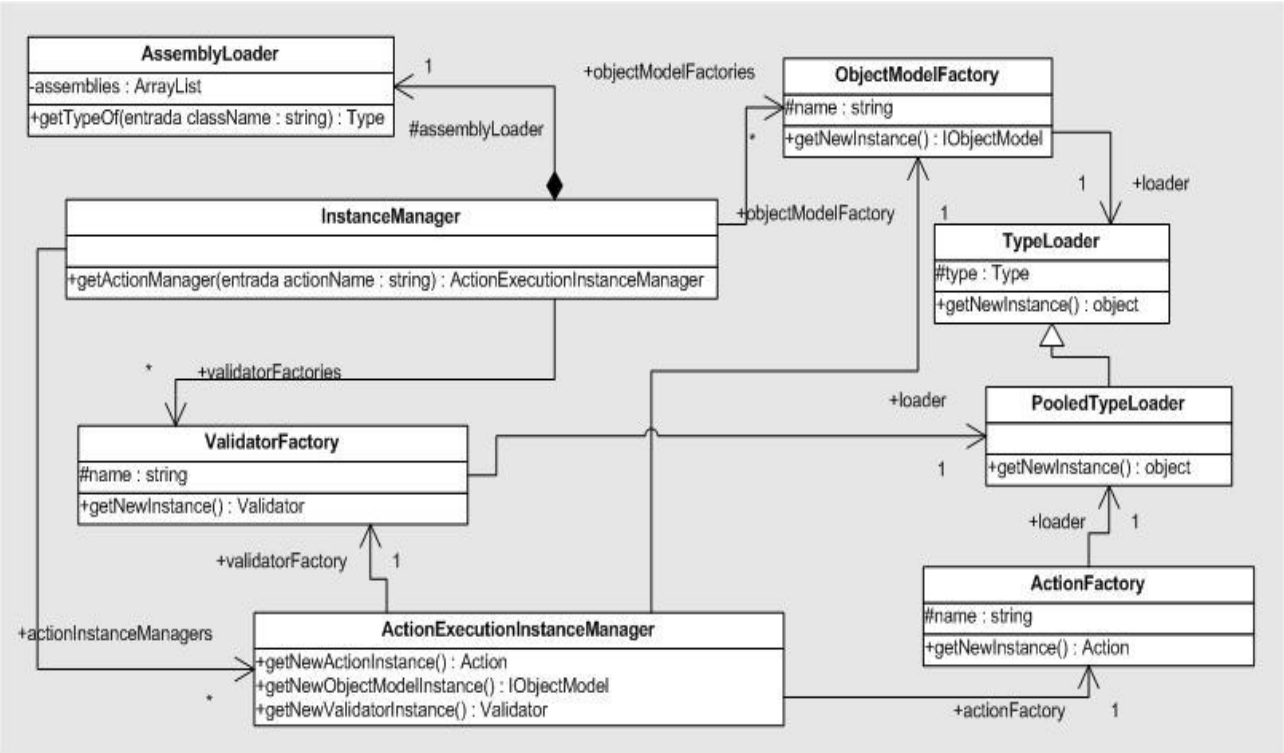


Figura 5: Componente de creación de instancias

Una vez que se posee el **ActionInformation** adecuado, se agrega a la tabla de manejadores de instancias una nueva instancia de **ActionExecutionInstanceManager** con todo lo que este objeto incluye:

- Una referencia a un **ActionFactory** encargado de crear mediante reflexión instancias de la clase de acción correspondiente para la acción especificada por el manejador, el cual es creado específicamente para esta instancia.
- Una referencia a un **ObjectModelFactory**, al cual se le delega la creación, también mediante reflexión, de instancias de los ObjectModels relacionados con la acción. Esta referencia puede ser obtenida de la colección de **ObjectModelsFactory** que posee el propio **InstanceManager**.
- Una instancia de **ValidatorFactory** quien se encargará de la creación del validador indicado para esta acción. De manera similar al caso de los ObjectModel, esta instancia de **ValidatorFactory** puede ser obtenida desde la colección de Validators que posee el **InstanceManager**.

Una vez que se tiene la instancia de **ActionExecutionInstanceManager**, sea que haya sido creada y agregada a la colección de manejadores de instancias o que haya sido simplemente obtenida del mismo contenedor, ésta es entregada al módulo de ejecución de acciones. Más tarde, durante su utilización, este objeto recibirá las solicitudes de creación de objetos a través del siguiente protocolo:

- **getNewActionInstance()**: retorna una nueva instancia del tipo de acción específica contenida por el manejador de instancias.
- **getNewValidatorInstance()**: este método es llamado cuando la acción posee un validador y tiene como función retornar una instancia del validador indicado por su factory de validadores.
- **getNewObjectModelInstance()**: de la misma forma que con el validador éste método se ejecuta únicamente cuando el la acción utiliza un ObjectModel de entrada. Cuando es llamado, este método retorna una nueva instancia del ObjectModel especificado.

Cabe aclarar, que para respetar todas las ventajas provistas por los objetos, se hizo uso del polimorfismo con el objetivo que todas las distintas fábricas de objetos del modelo respeten el mismo mensaje de **getNewInstance()** para la creación de instancias y una variable **name** la cual

contiene el nombre de la clase que se debe crear. También a modo de entender mejor el modelo realizado, se puede en la figura 5 que todas las fábricas poseen un referencia a un `TypeLoader`, quien es realmente el encargado de hacer efectiva la creación de los objetos. Actualmente, el Framework posee dos formas diferentes de creadores:

- ***TypeLoader simple***: Cada vez que recibe el mensaje polimórfico `getNewInstance()` crea una instancia y la retorna.
- ***PooledTypeLoader***: Este tipo de `TypeLoader` posee un pool con objetos de la clase a crear que se encuentran sin ser utilizados y un valor que indica la cantidad máxima de instancias de esta clase que ese pool debe tener. De esta forma, ante un pedido de obtención de una instancia, este cargador intenta entregar un objeto que se encuentre en la colección, de manera de ahorrar el tiempo de la creación del objeto y únicamente ante la excepción de que no haya objetos en dicha colección se deberá efectuar realmente la creación y alocación en memoria de una instancia de la clase solicitada.

Aunque, tal cual se mencionó recién, se proveen actualmente solo dos modos de creación de objetos, el modelo se encuentra preparado para soportar otras formas diferentes de crear objetos sin complicar en absoluto a los demás elementos del módulo.

6.5 Comunicación con la vista

El módulo de comunicación con la vista es el componente del framework que se encarga de interactuar directamente con los componentes vistas provistos por el Framework .net o por el desarrollador. La tarea principal de este módulo es la de operar como una especie de puerta de entrada al Framework con el objetivo de que el propio Framework trabaje de la misma forma independientemente del tipo de vista con que se encuentre interactuando.

A su vez, otra tarea no menos importante que posee este componente de comunicación con vistas específicas es el de permitir que cualquier tipo de vista, a través de una mínima implementación pueda utilizar al Framework WMVC como modelo de programación de su sistema. Dicha implementación, la cual será vista en detalle más adelante, consigue, sin importar las enormes incompatibilidades existentes entre distintas interfaces, adaptar una interfaz concreta a la interfaz genérica que necesita el WMVC Framework para ejecutarse.

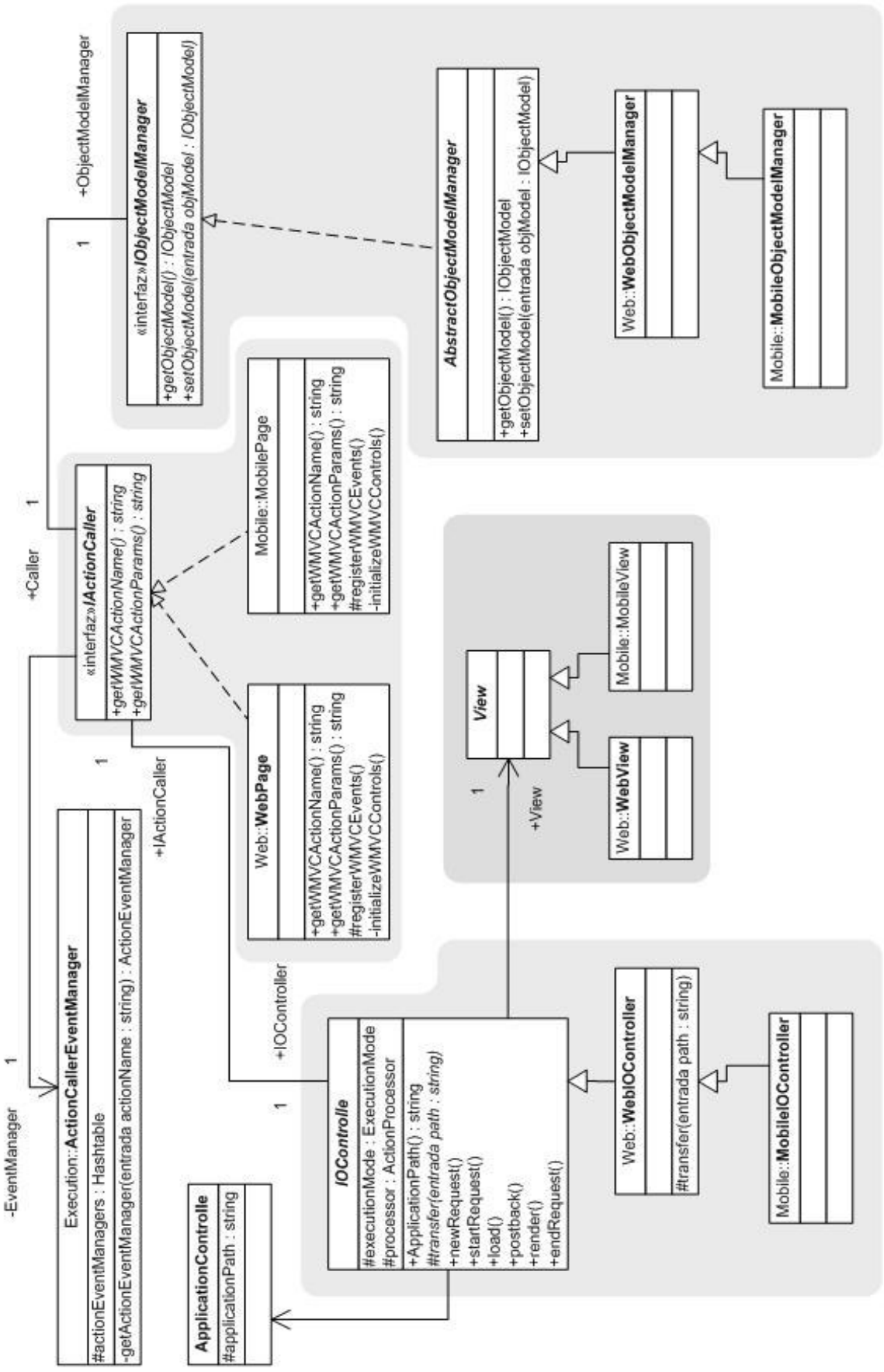


Figura 6: Módulo de interacción con la vista

Como ya se mencionó en varios tramos del documento actual, el WMVC Framework brinda actualmente soporte para dos tipos diferentes de vistas: Web y mobile, sin embargo, el framework fue desarrollado pensando en que los tipos de vistas soportados sean extensibles, configurables y adaptables. Para ello, este módulo fue diseñado e implementado con ese objetivo en mente. De esa manera, se consiguió abstraer ciertos mecanismos o formas que cada tipo de vista podría llegar a hacer de diferente manera:

- Distintos repositorios según el tiempo de vida de un object model en particular.
- Forma en que estos repositorios son accedidos para obtener y almacenar object models.
- Manera en la que cada vista interpreta el estado resultante de ejecutar una acción (redireccionamiento hacia otra vista)
- Mecanismo de llegada al servidor de los datos del cliente.
- Mecanismo de procesamiento en el servidor de la vista específica.

¿Cómo lograr generalizar todas las características mencionadas en este componente? Como primera medida para conseguir el objetivo se pensó que todas las interfaces gráficas debían implementar una interfaz de manera de generalizar sin proveer comportamiento, pero si un protocolo que permita interactuar genéricamente con ellas. Esta interfaz, **`WMVC.IActionCaller`**, posee:

- Propiedad **`IOController`**: obtiene la instancia de **`WMVC.IOController`** que se encargará de hacer de puerta de entrada y salida hacia el framework.
- Propiedad **`EventManager`**: obtiene un manejador de eventos utilizado para los eventos de **`PreActionExecute`**, **`PostActionExecute`** y **`PagePostActionExecute`**.
- Propiedad **`ObjectModelManager`**: posee un **`WMVC.IObjectModelManager`**, el cual estará encargado de interactuar con los distintos repositorios de cada vista de manera específica pero transparente para los objetos que se comuniquen con él.
- Mensaje **`getWMVCActionName()`**: la implementación de dicho mensaje debe retornar el nombre de la acción que se intentará ejecutar.
- Mensaje **`getWMVCActionParams()`**: dicha implementación deberá retornar un string con formato **`key=value`** y separado por el carácter **`&`** con la colección de parámetros de la acción que se intenta ejecutar.

¿Cómo subclasificar la clase `WMVC.IOController` para una vista propia específica? La clase `IOController` es quizás la clase más complicada que debe sobrescribirse para lograr que una vista se adapte al Framework WMVC debido a que debe responder a dos tipos diferentes de funciones:

- ***Funciones de entrada***, separadas en seis mensajes diferentes que marcan etapas dentro del procesamiento de un requerimiento WMVC. Estas etapas, ocurren todas durante la rutina de ejecución de la acción y cada una marca un evento o hecho en este proceso:
  - `newRequest()`: esto ocurre inmediatamente luego de que el requerimiento WMVC haya sido creado.
  - `startRequest()`: esta etapa representa al momento en que los demás objetos del modelo ya han sido creados pero el procesamiento del requerimiento WMVC todavía no ha comenzado.
  - `load()`: Indica que la carga de la vista actual ha sido creada, y todos sus controles inicializados.
  - `postback()`: representa el momento en que la vista se encuentra lista para correr cualquier tarea que necesite usar algún dato cargado en la misma.
  - `render()`: esta etapa permite acceder a los valores a mostrar en la vista inmediatamente antes de que éstos sean enviados o convertidos a la forma en que el cliente los debe mostrar.
  - `endRequest()`: éste último momento representa la finalización del requerimiento WMVC.

  Estas funciones de entrada permiten separar el proceso de la ejecución de una acción en diferentes pasos, fomentando directamente a la construcción de distintos modos de ejecución de acciones[2].
- ***La función de salida***, que involucra el posible redireccionamiento desde la vista actual hacia otra vista del sistema.

Con el modelo presentado, cada tipo diferente de interfaz gráfica, solamente necesita implementar la interfaz `WMVC.IActionCaller` y proveer las clases específicas de `View`, `IOController` y `IObjectModelManager` que soporten los requerimientos del Framework explicados anteriormente.

[2] ver sección 5.3.1 del presente documento: Modo de una acción.

Como se puede observar en la figura 6, los soportes para páginas web como para páginas web para aplicaciones móviles implementan esa interfaz mediante **WMVC.Web.WebPage** (heredando además de **System.Web.UI.Page**, clase base de las páginas web de ASP.NET) y **WMVC.Mobile.MobilePage** (extendiendo a la clase abstracta que representa a las páginas web creadas para dispositivos móviles: **System.Web.UI.MobileControls.MobilePage**) respectivamente.

Ambas interfaces, web y mobile, implementan esa interfaz retornando cada una sus instancias específicas de **IOController** (**WMVC.Web.WebIOController** y **WMVC.Mobile.MobileIOController** respectivamente), y de **IObjectModelManager** (**WMVC.Web.WebObjectModelManager** y **WMVC.Mobile.MobileObjectModelManager**), los cuales implementan los procesos necesarios para adaptarse a las necesidades del framework WMVC.

6.6 Ejecutor de acciones

El módulo ejecutor de acciones es el último de los módulos del Framework Core del WMVC Framework. Éste módulo tiene entre sus funciones efectuar los preparativos necesarios previo a la ejecución de una acción, luego llevar a cabo la ejecución real de la acción para finalmente realizar las tareas posteriores a la ejecución las que incluyen comunicarse con la vista para proveerle los datos resultantes y el estado en que la aplicación quedó luego de la acción.

El ejecutor de acciones comienza su ejecución cuando el módulo de interacción con la vista le informa que ha recibido un requerimiento WMVC que debe ser procesado. Para ello, el **IOController** usado para procesar el requerimiento crea una instancia de **ActionProcessor** y le indica que prepare el modo de ejecución adecuado para ejecutar la acción solicitada. Este **ActionProcessor** determina que modo de ejecución específico debe utilizar a través de información que le solicita al **ApplicationController** que posee la aplicación, para que en forma inmediata se lleve a cabo:

- La creación de una instancia del modo de ejecución determinado
- La instanciación de un **WMVC.Execution.ActionContext**, el cual representará el contexto en el cual la acción se ejecutará.

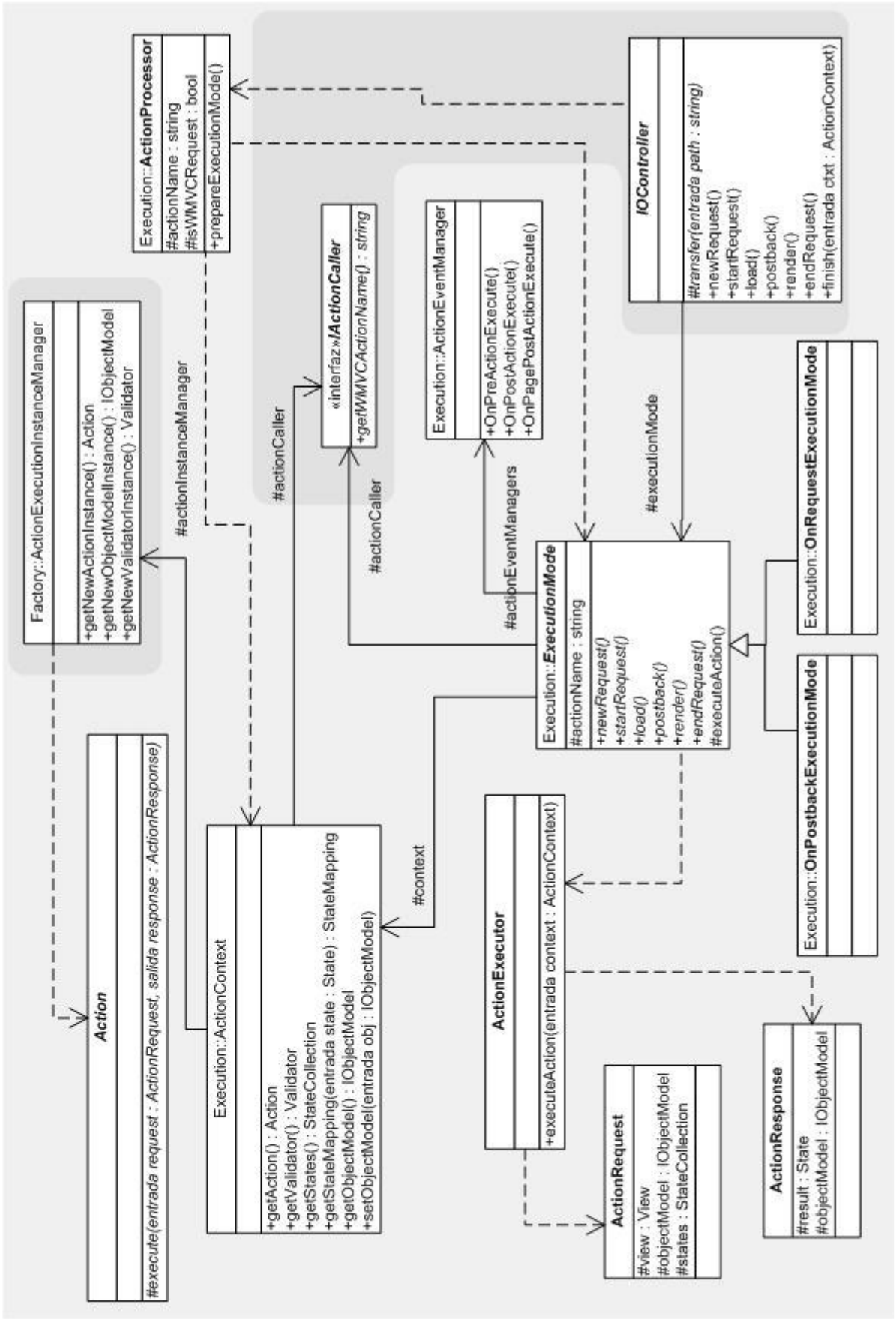


Figura 7: Módulo Ejecutor de acciones

Una vez obtenida la instancia del **ExecutionMode** a utilizar para la ejecución, el **IOController** comienza a enviarle mensajes a éste indicándole las diferentes etapas por las que va pasando la vista específica que produjo el requerimiento y de acuerdo a ello, el **ExecutionMode** actúa.

Cada implementación de **ExecutionMode** actuará diferente frente a cada uno de los seis pasos que debe atravesar a lo largo de una solicitud WMVC, sin embargo, independientemente de las diferencias conceptuales de cada uno de ellos, todas en algún momento deberán ejecutar la acción solicitada. Será allí entonces cuando cada modo de ejecución deba llamar al método **executeAction()** implementado en la clase abstracta **WMVC.Execution.ExecutionMode** para que lleve acabo la ejecución de la acción. Este método simplemente instancia un nuevo **WMVC.Execution.ActionExecutor** y le indica, también vía el mensaje **executeAction()** que debe ejecutar la acción utilizando el **ActionContext** que poseía el modo de ejecución.

Es así entonces, como sin importar específicamente el modo de ejecución de la acción, el **ActionExecutor** se hace cargo del control para luego llevar a cabo los siguientes pasos con el objetivo de ejecutar realmente la acción solicitada por el cliente:

1. Le pide al contexto de la acción a través del método **getObjectModel()** una instancia del Object Model de entrada configurado para la acción a ejecutar. Para ello, el contexto debe o entregar uno creado si es que ya existe uno en el repositorio específico que se indicaba para esta acción o crear uno nuevo
2. Le solicita al contexto un **Validator** para la acción a ejecutar, e inmediatamente, si es que existe lo evalúa. En caso de que la validación falle entonces todo el proceso de ejecución de la acción es abortado y el transcurso normal de un requerimiento ASP.NET es retomado.
3. Informa a la instancia de **ActionEventManager** que controla los eventos del proceso de ejecución actual que dispare el evento **PreActionExecute**.
4. Crea una instancia de **ActionRequest** la cual será utilizada durante la ejecución de la acción. A este **ActionRequest** se le cargan los posibles estados en que puede terminar la acción así como también se le carga el tipo de vista que produjo el requerimiento y el object model de entrada.
5. Le pide al contexto de la acción que le provea de una instancia de la acción a ejecutar mediante el método **getAction()**.

6. Ejecuta la acción recién creada enviándole el **ActionRequest** como información y esperando le sea retornada una instancia de **ActionResponse**. En este paso, es donde, en forma completamente transparente actúa el módulo de transporte entre capas, de manera de poder ejecutar la acción en diferentes formas en cuanto a aislamiento se refiere.
7. Inmediatamente posterior a la ejecución, nuevamente se comunica con el manejador de eventos de la acción, el **ActionEventManager**, para informarle que dispare el evento **PostActionExecute**.
8. Del **ActionResponse** resultante de la ejecución obtiene el Object Model de salida, el cual es entregado al contexto para que lo almacene correctamente comunicándose con el **IActionCaller** que generó la solicitud y con su correspondiente manejador de object models.
9. Finalmente, le comunica al **IOController** que inicialmente comenzó el proceso sobre el estado resultante de la ejecución de la acción (mediante el mensaje **finish()**) para que este realice el redireccionamiento necesario hacia la vista correspondiente. A partir de ese estado resultante, el **IOController** específico que lo reciba aplicará su forma particular de realizar el redireccionamiento. Es decir, por ejemplo, el **WMVC.Web.WebIOController** que se encarga de funcionar como controlador de entrada/salida para las aplicaciones web realiza el redireccionamiento hacia otra visto mediante un **Server.Transfer()** mientras que por ejemplo un controlador WMVC para aplicaciones de escritorio podría efectuar algún mecanismo mas complejo de redireccionamiento como podría ser la implementación de alguna interfaz con redireccionadores o similar.

Ahora, que ya hemos explicado más en detalle el funcionamiento del módulo de ejecución de acciones, podemos entender claramente el rol de que cumple cada uno de los elementos que lo componen:

ActionProcessor
- Determina en que modo se debe ejecutar la acción.
- Crea el Contexto.
- Crea el modo de ejecución adecuado.

ExecutionMode
- Cuando una implementación específica de **ExecutionMode** le envía un **ExecuteAction()** para que lleve a cabo la ejecución de la acción entonces crea un **ActionExecutor** y lo ejecuta.

	• Cuando llega al momento del postback, dispara el evento **`PagePostActionExecute`**.
`ActionContext`	• Proveerle al **`ActionExecutor`** de todos los objetos que necesita para su ejecución, como son la acción a ejecutar, el object model, el validador, los states, y hasta el **`IActionCaller`** que generó la solicitud. • Comunicarse con un **`ActionExecutionInstanceManager`** del componente de creación de objetos para que este le provea las instancias de los objetos que debe crear.
`ActionExecutor`	• Interactuar con el contexto para solicitarle datos de entrada a la acción • Llevar a cabo la ejecución de la acción • Interactuar con el contexto una vez ejecutada la acción para informarle de los resultados.
`Action`	• Proveer una interfaz para que las subclases se comuniquen con el modelo lo hagan sin preocuparse por la comunicación con los otros componentes del módulo.
`ActionRequest`	• Actuar como un contenedor en la transmisión de los datos hacia el modelo.
`ActionResponse`	• Actuar como un contenedor de los resultados de la acción hacia la vista.

Capítulo 7: La biblioteca de Controles

7.1 Introducción a la biblioteca de controles

Uno de los componentes más importantes del framework es la biblioteca de controles. Este componente es la parte del framework que está más relacionada con la vista de las páginas web: se encarga de la forma de mostrar los datos y de dónde y como obtenerlos.

La idea de la biblioteca de controles provista por el WMVC Framework es, simplemente, proveer un conjunto de herramientas que permitan al diseñador web diseñar, escribir y probar la página sin la necesidad de conocer ningún lenguaje de programación. Además, no necesitará que un desarrollador inserte código dentro de su página para poder mostrar los datos del modelo.

La tarea de la biblioteca de controles, o mejor dicho, de todos y cada uno de los controles WMVC es poder mostrar los datos del modelo usando un control o tag en HTML puro, y no mediante código del framework agregado a la página web donde el dato del modelo deba ir.

Con la utilización de esta biblioteca de controles, se consigue una gran aceleración en los tiempos de desarrollo de las aplicaciones web dado que la tarea que antes involucraba a dos personas y con subtareas casi completamente separadas (mientras el diseñador definía la forma en que los datos se mostraban, el desarrollador tenía que codificar la lógica para obtener los mismos en la propia página web), ahora solo requería del diseñador y de una buena documentación que le especificara al mismo lo que había que mostrar..

7.2 División de la biblioteca de controles

La biblioteca de controles se encuentra dividida en dos tipos de controles:

- ***Controles lógicos o de control de flujo***: Estos controles son utilizados para evitar codificación de control de flujo y de validación tanto sea en la página web (.aspx) mediante la inserción de tags `runat="server"` como en su página de atrás (aspx.cs o aspx.vb) cuando generalmente en el `pageLoad()` se cargan los objetos dinámicos.

- ***Controles visuales***: Este tipo de controles extienden a los controles definidos por el framework de ASP.NET. La extensión se basa principalmente en que los mismos brindan la posibilidad de especificar un valor dinámico para mostrar a partir de un objeto.

7.2.1 Controles lógicos del WMVC Framework

Así como los controles visuales del WMVC Framework agregan un interesante concepto a los controles web provistos por el .NET Framework, estos controles lógicos explicados aquí también introducen un concepto novedoso para los desarrolladores ASP.NET.

Uno de los más importantes beneficios que provee el framework WMVC es quizás una idea aplicada por primera vez en el framework para aplicaciones web en java, el Jakarta Struts: la presentación de un conjunto de tags lógicos que permitían obviar cierta codificación en java sobre validaciones e iteraciones.

¿Que lograba Jakarta Struts con esos controles lógicos? Mediante la utilización de estos tags, Struts conseguía no solo que el diseñador pueda realizar la vista sin problemas sino que además lograba disminuir los tiempos de desarrollo ya que al haber menos código fuente del lenguaje de programación, menos errores ocurrían.

```
<table width="70%" border="1" >
    <tr>
        <th> Código
        <th> Nombre
    </tr>

    <logic:iterate name="facultad" id="car" property="carreras">
    <tr>
        <td> <bean:write property="codigoCarrera" name="car" /> </td>
        <td> <bean:write property="descripcion" name="car" /> </td>
    </tr>
    </logic:iterate>
</table>
```

Figura 1: Código JSP/Struts utilizando tags lógicos

En la figura 1 se puede observar que para la carga de una tabla con carreras, utilizando el framework de Struts, no fue necesario ni una sola línea de código fuente JAVA. En el ejemplo se pueden divisar los tags lógicos como el **`<logic:iterate>`** y también tags visuales como el **`<bean:write>`**, otro tag definido por Struts que escribe en la salida el valor indicado por la combinación name + property.

A partir de esa idea, y de los buenos resultados que tuvo esa idea en el desarrollo de aplicaciones web, se decidió que el WMVC Framework también los defina, lo que parece traer, a primera vista más soluciones y beneficios que para Java.

```
<table width="70%" border="1">
    <tr>
        <th> Código
        <th> Nombre
    </tr>

    <%
  Vector carreras = (Vector) session.getValue("facultad.carreras");
  for (int index=0; index < carreras.size(); index++)
  {
    Carrera car = (Carrera) carreras.elementAt(index);
    %>
    <tr>
        <td ><%= car.codigocarrera() %> </td>
        <td ><%= car.descripcion() %> </td>
    </tr>
    <%
  }
    %>
</TABLE>
```

Figura 2: Código JSP sin la utilización de Struts

¿Porque más beneficios? Porque la carga normal de una página JSP ocurre todo de una vez por lo que, por ejemplo una iteración para cargar una tabla se encuentra agrupada en un bloque de código JSP donde se define la tabla y se carga la misma. Ya vimos el ejemplo de la carga de una tabla con carreras utilizando Struts en la figura 1; en la figura 2 se puede visualizar el mismo ejemplo pero utilizando JSP únicamente, teniendo como clara desventaja la existencia de código Java no solo para realizar la iteración sino también para obtener los valores a mostrar.

```
<asp:Repeater ID="cargaTablaCarreras" Runat="server">
    <HeaderTemplate>
        <Table width="70%" Width="1" id="tableCarreras" runat="server">
            <Tr>
                <th> Código </th>
                <th> Nombre </th>
            </Tr>
    </HeaderTemplate>
    <ItemTemplate>
            <Tr>
                <td> <asp:Label ID="labelCodigo" Runat="server"> </asp:Label> </td>
                <td> <asp:Label ID="labelDescripcion" Runat="server"> </asp:Label> </td>
            </Tr>
    </ItemTemplate>
    <FooterTemplate>
        </Table>
    </FooterTemplate>
</asp:Repeater>
```

Figura 3: Declaración de una tabla en ASP.NET

En cambio, en ASP.NET esto se encuentra separado por múltiples lugares teniendo la desventaja de que es notablemente más complicado el control de la carga y definición de la misma. La separación que provee ASP.NET de la página html, con extensión .aspx para diferenciarlas de páginas web estáticas, del contenido dinámico de la misma, situado en la página de atrás, escrita en alguno de los lenguajes del framework .NET (para el caso de C# sería .aspx.cs), no siempre es una ventaja.

En las figuras 3 y 4 se puede ver el mismo ejemplo de la carga de la tabla con carreras de una facultad escrito en .NET. En la figura 3 se puede ver la estructura gráfica de la página, escrita totalmente con código HTML y pudiendo ser desarrollada por un diseñador gráfico únicamente. En ella se definen únicamente una tabla y su contenido. Además, se utiliza un control propio de ASP.NET, Repeater, cuya tarea es la de recorrer los elementos de un determinado datasource el cual es provisto en ejecución.

```
protected Repeater cargaTablaCarreras;

private void Page_Load(object sender, System.EventArgs e)
{
    //...

    this.cargaTablaCarreras.DataSource = (ArrayList) this.Session["facultad.carreras"];
    this.cargaTablaCarreras.DataBind();

    //...
}

private void InitializeComponent()
{
    this.Load += new System.EventHandler(this.Page_Load);
    this.cargaTablaCarreras.ItemDataBound +=
        new RepeaterItemEventHandler(cargaTablaCarreras_ItemDataBound);
}

private void cargaTablaCarreras_ItemDataBound(object sender, RepeaterItemEventArgs e)
{
    if ((e.Item.ItemType != ListItemType.Header) &&
        (e.Item.ItemType != ListItemType.Footer))
    {
        Carrera car = (Carrera) e.Item.DataItem;

        ((Label) item.FindControl("labelCodigo")).Text = car.codigoCarrera();
        ((Label) item.FindControl("labelDescripcion")).Text = car.descripcion();
    }
}
```

Figura 4: Cargado de una tabla en ASP.NET

La figura 4 muestra la página de atrás para el mismo ejemplo de la figura 3. Aquí, se muestra únicamente el código que interesa ver para el cargado de la página obviando todo el resto

del código fuente de esa clase. En la primera parte del código se define la variable instancia `Repeater cargarTablaCarreras`, la cual será la encargada de cargar el contenido de la tabla. En el método `initializeComponent()` definido por el framework .NET y llamado durante la creación de la página, se setea un manejador para el evento `ItemDataBound`, el cual se produce uno por cada elemento del datasource, más dos veces más: una antes de la colección y una después. Todo continúa en el `pageLoad()` donde el repeater recibe su datasource y se le hace `databind()`. Este mensaje provocará el disparado del evento `ItemDataBound` y llevará directo a la ejecución del manejador antes registrado. En dicho manejador se cargará, por cada elemento del datasource, la estructura definida en la página web.

Es claro que ASP.NET, a pesar de tener varias ventajas como la encontrarse todo debidamente modularizado y de ser enormemente extensible y mantenible, no posee ciertas facilidades que si tienen otras tecnologías. Mientras que en JSP a uno le alcanzaba con algunas líneas de código JAVA embebidas en la página web, con ASP.NET se debe programar, definir y controlar no solo la parte gráfica sino también el funcionamiento y llenado de la misma.

ASP.NET cuenta con la ventaja de querer simular para una aplicación web un funcionamiento similar al de una aplicación de escritorio, con su debida definición de controles y eventos ocurridos sobre los mismos. Sin embargo, muchas veces los desarrolladores necesitan utilizar las ventajas provistas por las aplicaciones web en cuanto a la simplicidad de las mismas para definir vistas y manejar eventos sobre las ellas.

Debido a las desventajas que tenía ASP.NET en algunas tareas como la que acabamos de mostrar y como consecuencia de que la idea del WMVC Framework es facilitar el desarrollo de aplicaciones web en la programación, era casi una obligación pensar una solución para estos problemas. Para ello, se agregó una colección de controles lógicos que permiten a los desarrolladores web que utilizan el framework obtener las ventajas de los programadores y diseñadores de otras tecnologías.

¿Como quedaría entonces con los controles lógicos de WMVC? Con la utilización de la biblioteca de controles del framework WMVC el código quedaría muy parecido al ejemplo de Struts, dado que toda la declaración y carga del contenido de la tabla se haría conjuntamente con los controles visuales del WMVC

```
<WMVC:Iterate object="facultad" property="carreras" item="car" runat="server">
    <WMVC:Header>
        <Table width="70%" border="1">
            <Tr>
                <th> Código </th>
                <th> Nombre </th>
            </Tr>
    </WMVC:Header>
    <WMVC:Item>
            <Tr>
                <td>
                    <WMVC:Label object="car" property="codigoCarrera"
                                Runat="server"> </WMVC:Label>
                </td>
                <td>
                    <WMVC:Label object="car" property="descripcion"
                                Runat="server"> </WMVC:Label>
                </td>
            </Tr>
    </WMVC:Item>
    <WMVC:Footer>
        </Table>
    </WMVC:Footer>
</WMVC:Iterate>
```

Figura 5: Ejemplo de una tabla con los controles WMVC

En el ejemplo de la figura 5 se puede ver como, de manera muy parecida a Struts, el diseñador puede, además de definir la forma gráfica de la vista, cargar el contenido de la misma. En este ejemplo, el mismo que se efectuó para las otras tecnologías, se puede ver el uso del control lógico **`<WMVC:Iterate>`** el cual no hace ni mas ni menos que iterar sobre una colección especificada, dejando visible al diseñador al elemento actual del recorrido especificado por la propiedad **`item`** del mismo tag. Así, dentro del alcance de dicho tag, o sea todo lo que en la estructura de árbol de HTML se encuentre dentro del tag, cualquiera que tag que haga referencia a **`car`** estará indicando a un elemento de la colección especificada en el **`<WMVC:Iterate>`**.

7.2.2 Controles Visuales del WMVC Framework

De la misma manera que los controles lógicos, los controles visuales del WMVC Framework permiten disminuir notablemente la cantidad de código fuente a escribir y a consecuencia de eso también la disminución de la cantidad de errores y del tiempo de testeo en el proceso de construcción de aplicaciones web con ASP.NET.

¿Que es lo nuevo que traen estos controles web? La novedad de estos controles visuales es su posibilidad de cargarse dinámicamente a partir de cierta meta-información establecida al momento del diseño.

¿De que manera se logra eso? Quizás la manera más fácil de explicar esta ventaja sea mediante un ejemplo. Supongamos que un sistema necesita una pantalla de modificación de los datos personales de un usuario existente. En ella, se puede ver entre los tantos datos que contiene el nombre de usuario de la persona en modo solo lectura debido a que no se puede cambiar, luego el nombre de la persona en un campo de texto y finalmente un checkbox que indica si la persona es o no mayor de edad.

```
<tr>
    <td width="30%"> Nombre de Usuario: </td>
    <td> <WMVC:Label object="user" property="username" Runat="server"> </WMVC:Label> </td>
</tr>
<tr>
    <td width="30%"> Nombre Completo: </td>
    <td> <WMVC:TextBox object="user" property="fullName" Runat="server"> </WMVC:TextBox> </td>
</tr>
<tr>
    <td width="30%">
        <WMVC:CheckBox object="user" property="over18" Runat="server"> </WMVC:CheckBox>
    </td>
    <td> Es Mayor de Edad </td>
</tr>
```

Figura 6: Ejemplo del uso de controles visuales del WMVC Framework

En la figura 6 se puede ver la sección de la páginas .aspx que utilizando los controles del Framework define la pantalla antes mencionada. En ella, se utilizan tres controles **`<WMVC:Label>`**, **`<WMVC:TextBox>`** y **`<WMVC:CheckBox>`** a los cuales se les indica de que objeto y propiedad deben cargar sus valores y de esa manera, en el momento de renderizado de la página, cada control obtiene el valor de la combinación de **`object.property`**.

```
<tr>
    <td width="30%"> Nombre de Usuario: </td>
    <td> <asp:Label Runat="server" ID="labUsername"> </asp:Label> </td>
</tr>
<tr>
    <td width="30%"> Nombre Completo: </td>
    <td> <asp:TextBox Runat="server" ID="txtFullName"> </asp:TextBox> </td>
</tr>
<tr>
    <td width="30%"> <asp:CheckBox Runat="server" id="chkOver18"></asp:CheckBox> </td>
    <td> Es Mayor de Edad </td>
</tr>
```

Figura 7: El mismo ejemplo anterior pero con controles ASP.NET (Web)

Esta posibilidad de cargar dinámicamente los valores de los objetos era una restricción importante en el framework ASP.NET debido a que los desarrolladores podían únicamente hacerlo mediante el uso de reflexión.

¿Cual es la ventaja entonces de los controles WMVC? Las ventajas de estos controles contra los controles provistos por el Framework .NET son varias:

- Acceder a los valores de las propiedades de un objeto sin que el programador necesite escribir código fuente.
- La única forma de cargar los valores con los controles de ASP.NET es, como recién se mencionó, mediante reflexión y esto no es ni fácil de programar ni código fuente legible.
- Se logra una completa separación entre el diseño de la página web y la programación de atrás permitiendo una división total de los roles.

En las figuras 7 y 8 se visualiza el mismo ejemplo de la página de modificación de los datos personales de un usuario registrado en el sistema pero realizado con los controles de ASP.NET. En la figura 7 se puede ver la página web en la cual se utilizan tres controles donde en todos ellos se define únicamente un identificador de manera que pueda ser utilizado y cargado en la página de atrás.

```
protected Label labUsername;
protected TextBox txtFulllName;
protected CheckBox chkOver18;

private void Page_Load(object sender, System.EventArgs e)
{
    //...

    this.labUsername.Text = user.Username;
    this.txtFulllName.Text = user.FullName;
    this.chkOver18.Checked = user.over18;

    //...
}
```

Figura 8: El mismo ejemplo anterior pero con controles ASP.NET (Página de atrás)

Mientras tanto, en la figura 8 se ve el código C# de la página de atrás. En la parte superior se declaran como variables instancias de la clase los mismos controles que se definieron en la página web. Luego en el `Page_Load()` se realiza la asignación de ellos con las mismas propiedades que se habían definido en los controles WMVC del ejemplo anterior.

7.3 Descripción de los controles lógicos

Una vez ya definido los controles lógicos del WMVC Framework, ahora pasaremos a enunciar ciertas características más particulares de cada uno de los tres grupos básicos en que ellos se dividen: Validadores, Comparadores e Iteradores.

7.3.1 Controles lógicos Validadores

Este tipo de controles permite, mientras se genera la página web hacer cambios gráficos como podría ser tomar la decisión de mostrar o no un determinado elemento. Estos validadores verifican principalmente la presencia o ausencia de ciertos datos en el contexto de la página.

Para entender mejor realizaré un ejemplo donde es de gran utilidad este tipo de controles. Supongamos el mismo caso usado en las secciones anterior: la carga de la tabla con carreras de una facultad, pero con un nivel de especificación más detallado. Supongamos entonces que la tabla que antes tenía dos columnas: una para el código de la carrera y otra para la descripción, ahora se agrega una tercer columna, la cual contiene el nombre del titulo intermedio siempre y cuando éste no sea nulo. En la figura 9, se puede ver el código, únicamente de la tercera columna con el uso de los controles de WMVC.

```
<td>
    <WMVC:IsNotNull object="car" property="tituloIntermedio">
        <WMVC:Label object="car" property="tituloIntermedio.descripcion"
                    Runat="server"> </WMVC:Label>
    </WMVC:IsNotNull>
</td>
```

Figura 9: Ejemplo de un control lógico validador

Existen además, otros controles validadores que se utilizan de manera similar al control `isNotNull` mostrado en la figura anterior

Control	*Condición a evaluar*
`isNull`	Valida si el valor obtenido es nulo
`isNotNull`	Valida si el valor obtenido es distinto de nulo
`isEmpty`	Valida si el valor obtenido es una colección sin elementos o

	un string vacío.
`isNotEmpty`	Valida si el valor obtenido es una colección con al menos un elemento o un string con uno o mas caracteres.

Con el uso de estos controles, el diseñador puede ahora controlar también los valores a mostrar para casos excepcionales.

7.3.2 Controles lógicos Comparadores

Al igual que los controles enunciados en la sección anterior, este tipo de controles permite, durante la construcción de la página web hacer cambios gráficos dependiendo de una comparación entre dos valores.

Con el objetivo de que el lector pueda comprender las ventajas de este tipo de control dejaremos de lado el ejemplo últimamente utilizado y pasaremos a presentar uno más simple y fácil: Supongamos el home de un sitio donde en caso de que el usuario sea mayor de edad se le muestre contenido adicional. En dicho ejemplo se puede ver que compara si `user.age` es mayor a 18 mediante el uso del control `<WMVC:GreaterThan>`.

```
<WMVC:GreaterThan object="user" property="age" value="18">
    <H2> Contenido visible unicamente por personas mayores de edad </H2>
    <...

    ...>
</WMVC:GreaterThan>
```

Figura 10: Ejemplo de un Control lógico Comparador.

Así como en el ejemplo anterior se utilizó el `GreaterThan`, existen los siguientes tipos de comparadores en el Framework WMVC:

Control	*Condición a evaluar*
`EqualsTo`	Verifica por igualad
`NotEqualsTo`	Verifica por desigualdad
`LessThan`	La comparación es realizada por menor estricto
`LessOrEqualTo`	La comparación es realizada por menor o igual

`GreaterOrEqualTo`	La comparación es realizada por mayor o igual
`GreatherThan`	La comparación es realizada por mayor estricto

Todos ellos evalúan la condición con una combinación de object y property como primer argumento y un como segundo.

7.3.3 Controles lógicos Iteradores

La idea de los controles iteradores quizás es la que hasta este momento se entendió mejor dado que la mayoría de los ejemplos presentados incluían al control de iteración **<WMVC:Iterate>**. Sin embargo, existe otro control que también itera **<WMVC:IterateWhile>** y además también existen distintas posibilidades dentro de cada uno de ellos.

Para el caso particular del **iterate** existen diferentes combinaciones en sus propiedades que llevan a distintas posibilidades de iteraciones:

- Si se especifica únicamente la colección entonces se realizará un recorrido en orden de la misma renderizando por cada elemento el **<WMVC:ITEM>** correspondiente.
- Si además de especificarse la colección se setea un valor de comienzo y uno de fin (mediante las propiedades **Start** y **End** respectivamente) entonces el recorrido de la colección se hará desde la posición indicada en **Start** hasta la que se especificó en **End**.
- Si únicamente se define un número de inicio y uno de fin, entonces el resultado sería la iteración desde una posición hasta la otra comportándose de manera parecida a un bloque **for** como se puede ver a continuación

```
for (int i = inicio;  i < fin; i++)
```

Independizándonos de las distintas combinaciones del tag **<iterate>**, todas ellas llevan tres controles opcionales en su interior.

- **<WMVC:Header>**: Este control es utilizado para mostrar información de cabecera, o previo a la ejecución de la iteración. Sirve para definir los encabezados de una tabla o para poner por ejemplo un control de paginación y ciertas funciones propias del contenido a mostrar.
- <**WMVC:Item>**: Dentro de este control se definen los controles que se utilizarán en la repetición de cada elemento de la colección.

- **<WMVC:Footer>**: Aquí, se puede definir cierta información propia de la tabla como controles de paginación o muestra de totales de alguno de los campos de la colección recorrida.

7.4 Descripción de los Controles Visuales

Los controles visuales del Framework WMVC también se encuentran divididos en dos debido al objetivo que tiene cada uno de los grupos:

- Controles de Datos
- Controles de Acción

7.4.1 Controles de Datos

El objetivo principal de este tipo de controles visuales es simplemente mostrar información de manera gráfica. Estos controles, quizás sean los más parecidos a los provistos por el Framework de .NET y además, la mayoría de estos se encuentran relacionados íntimamente con sus similares en el .NET. La única diferencia que se presenta entre un control WMVC particular y su correspondiente web control de .NET es, como ya lo habíamos mencionado antes, la posibilidad de cargar sus datos desde propiedades y objetos definidos en la propia página web logrando:

- Que el tamaño de la página de atrás sea más pequeño debido a que muchas cosas que antes se hacían allí ahora son hechas automáticamente por los propios controles.
- Menor cantidad de errores en la programación de la página de atrás
- Mayor separación entre los diseñadores y los desarrolladores.
- Mayor claridad en el código fuente de la página de atrás no solo por la existencia de menos código sino también porque en ella queda únicamente funcionalidad propia de los eventos de la página.

A continuación se enumeran la lista de controles de datos provistos por el WMVC Framework donde al lado de cada uno de ellos se menciona a que control web del .NET extienden.

Control	*Control web base del .NET*
Label	**System.Web.UI.WebControls.Label**
TextBox	**System.Web.UI.WebControls.TextBox**

`Hidden`	`System.Web.UI.HtmlControls.HtmlInputHidden`
`HyperLink`	`System.Web.UI.WebControls.HyperLink`
`CheckBox`	`System.Web.UI.WebControls.CheckBox`
`RadioButton`	`System.Web.UI.WebControls.RadioButton`
`DropDownList`	`System.Web.UI.WebControls.DropDownList`
`ListBox`	`System.Web.UI.WebControls.ListBox`

7.4.2 Controles de Acción

El .NET Framework utiliza para la construcción de aplicaciones web tres controles bases para generar eventos en el servidor: `Button`, `LinkButton` e `ImageButton`. Estos controles, al ser clickeados ejecutan una función javascript, la cual es muy importante para el funcionamiento de ASP.NET llamada `__doPostback()` cuya tarea es la de almacenar cierta información de la página web y luego enviar el formulario de la misma hacia el servidor.

Dado que el funcionamiento del WMVC Framework se encuentra basado principalmente en acciones, entonces los controles de acción del Framework realizan algo muy parecido a lo que hace el Framework .NET: Cuando son clickeados realizan una llamada a la función javascript `__doWMVCPostBack()` con información de la acción que se desea ejecutar. Esta función almacena esa información y luego delega responsabilidades en la función `__doPostback()` del .NET para el envío del formulario al servidor.

Existen entonces dos diferencias importantes entre los controles del .NET y los de acción definidos por el WMVC Framework:

- ***En comportamiento***: los controles del .NET se encuentran totalmente orientados a eventos proporcionados por los controles mientras que los del WMVC, utilizan esa idea para enviar la información de la acción que se debe ejecutar
- ***En información***: Mientras que los controles del .NET no envían ninguna información hacia el servidor de manera visible (está hecho a escondidas en la función `__doPostBack()`), los controles del WMVC envían explícitamente, mediante la propiedad `action,` información sobre la acción a ejecutar siendo esto algo bastante visible y claro.

```
<tr>
    <td colspan="2">
        <WMVC:Label object="licence" property="text"
                runat="server" ID="labText"></WMVC:Label>
    </td>
</tr>
<tr>
    <td>
        <WMVC:Button text="Aceptar" action="acceptLicence"
                     runat="server" id="btnAceptar"></WMVC:Button>
    </td>
    <td>
        <WMVC:Button text="Rechazar" action="declineLicence"
                     runat="server" id="btnRechazar"></WMVC:Button>
    </td>
</tr>
```

Figura 11: Ejemplo de un control de acción

En la figura 11 se puede ver un ejemplo de estos controles. Suponemos una página web que necesita mostrar información de una licencia y dos botones: uno para aceptar la misma y otra para rechazarla. Para ambos controles **`<WMVC:Button>`** se puede ver que utilizan la propiedad action para indicar que acción se necesita ejecutar en el servidor.

Capítulo 8: Herramientas que complementan al WMVC Framework

8.1 Breve introducción a las herramientas complementarias

Si bien uno de los objetivos del Framework es utilizar el mismo para facilitar la programación de las aplicaciones, tanto web como mobile como de cualquier otro tipo de aplicación para el cual se implementen las clases necesarias para comunicarse con el Framework y agregar herramientas complementarias a la programación no siempre faciliten las cosas, el WMVC Framework introduce dos nuevas herramientas.

Aunque ninguna de las dos aplicaciones tenga que ser obligatoriamente usada para poder utilizar el Framework sabemos perfectamente que el uso de ellas tendrá un resultado positivo en el desarrollo de las aplicaciones.

Tanto el ***Configuration Tool*** (ver 8.2) como el ***Source Code Writer*** (sección 8.3) son muy fáciles de usar y permiten que, quizás las dos tareas más complicadas o repetitivas y consumidoras del tiempo de desarrollo que se tienen como consecuencia de utilizar el Framework, puedan ser realizadas en un entorno cómodo y amigable al programador.

A continuación se presentarán las dos herramientas, comentando brevemente sus funcionalidades y enunciando las ventajas de utilizarlas frente a realizar las mismas acciones sin ellas.

8.2 Configuration Tool

Cualquier aplicación que utilice el WMVC Framework debe crear al menos uno o varios archivos de configuración en donde se especifiquen las acciones, los objectModels utilizados por las mismas, los estados en los que puede quedar el sistema o que se pueden obtener como resultado de una acción y también información sobre hacia que vista se debe redireccionar el sistema dependiendo del tipo de solicitud que se haya generado la ejecución de una acción. Para aplicaciones pequeñas, la generación y mantenimiento de estos archivos no sería un problema ya

que no existirían demasiados elementos que se deban configurar, pero a medida que el tamaño del sistema aumenta la complejidad de la escritura de la configuración se torna más complicada.

Aparte de que la complejidad se encuentra directamente relacionada con el tamaño del sistema que se desea configurar, existe otro problema, y es que los archivos de configuración deben ser legibles tanto para una persona como por una computadora. Para ello, se eligió el formato XML que satisface esa necesidad de fácil lectura tanto por humanos como por máquinas, tal cual se explicó en el capitulo anterior. Sin embargo, los archivos de este tipo no son ni rápidos ni fáciles de escribir ya que se deben respetar ciertas estructuras y normas que hacen que un mínimo error de tipeo deje inválido al archivo, teniendo como consecuencia inmediata errores al momento de ser cargados por el WMVC Framework.

Estos errores de archivos de configuración inválidos generan funcionamiento incorrecto, o directamente no funcionamiento, de la aplicación que se está construyendo retrasando los tiempos de desarrollo y testeo.

El ***Configuration Tool*** permite leer y escribir los archivos de configuración a través de una aplicación GUI. La ventaja de esta aplicación es básicamente permitir que el usuario del framework se abstraiga del formato específico de los archivos de configuración y se focalice únicamente en generar la estructura de configuración adecuada para su sistema.

Con la utilización del Configuration Tool se logra:

- Que el programador no necesita en ningún momento interactuar con los archivos de configuración.
- Evitar problemas de archivos de configuración no válidos
- Que el encargado de configurar el sistema se dedique exclusivamente a ello y no a preocuparse por generar correctamente los archivos.
- Que la persona a cargo de la configuración pueda generar la misma de manera veloz y eficaz.
- Que la configuración sea realmente legible tanto por el sistema, que entiende los archivos de configuración como el humano que interactúa con ésta aplicación.

8.3 Funcionamiento de la herramienta "Configuration Tool"

El Configuration Tool es una herramienta presentada en forma de aplicación de escritorio que permite, a partir de la carga del archivo de configuración del paquete principal `WMVC.config` mantener, controlar y modificar toda la configuración del sistema, incluyendo no solo el archivo de configuración principal sino también todos los que se desprenden de él, configurados como subpaquetes.

Una vez cargado un proyecto en la aplicación, la ventana principal muestra sobre la izquierda la estructura de árbol de los paquetes del sistema mientras que a la derecha se tiene la posibilidad de setear las librerías a usar por el Framework además de también tener funciones específicas sobre los paquetes (creación, edición, borrado, etc.).

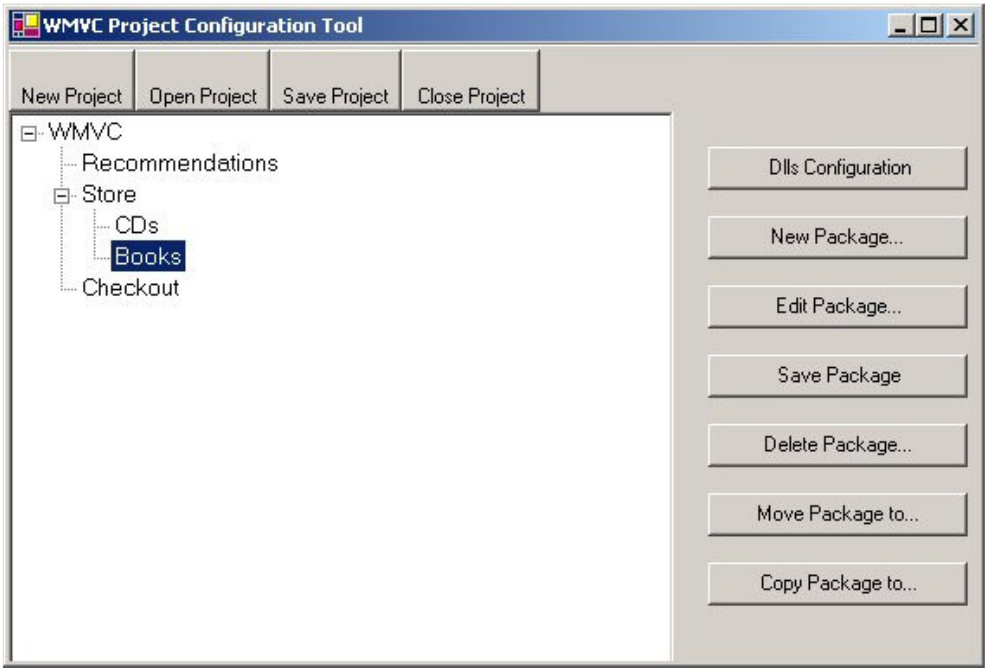


Figura 1: Vista de la configuración del CD Store en el Configuration Tool

En la figura 1 se presenta la ventana principal del Configuration Tool con el proyecto del CD Store cargado en donde se puede apreciar la jerarquía de paquetes que refleja exactamente la misma relación definida entre los archivos de configuración.

Esta herramienta genera los archivos de configuración de la mejor manera posible, haciendo que, por ejemplo, las librerías del usuario utilizadas por el Framework sean cargadas en el archivo de configuración principal y no en el sus subpaquetes, dado que ellas, tal cual se explica luego en el

Apéndice A, son independientes de la jerarquía de paquetes en que se puede estructurar el sistema. Para ello, independientemente del paquete que se encuentre seleccionado, presionando en el botón ***"Dlls Configuration"*** de la derecha se despliega una ventana (Figura 9.2) con la lista de las librerías a setear en el archivo principal de configuración **`wmvc.config`**.

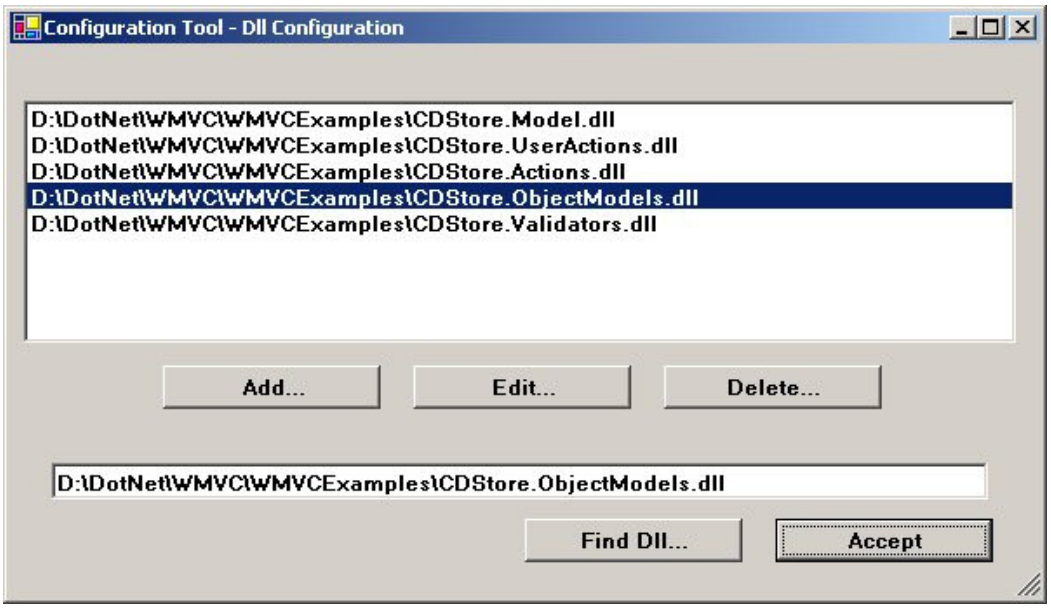


Figura 2: Configuración de las librerías a utilizar en el sistema

En la ventana principal del Configuration Tool se divisa a la derecha una lista de acciones a efectuar sobre los paquetes del sistema. Seguramente la más compleja de todas ellas sea la de modificar o crear un nuevo paquete dado que es en ese momento en el cual la persona encargada de generar la configuración necesita de toda la ayuda para cargar de manera correcta sus acciones, objectModels, estados, vistas, validadores, y demás información.

En la figura 3, se ve un ejemplo claro sobre la edición de un paquete: Se muestra el paquete **`Store.CDs`** obtenido desde el archivo de configuración **`Store.CDs.config`**. La ventana se encuentra dividida en tres partes:

- ***Información general del paquete***: Ubicado en la parte superior de la ventana, se muestra el nombre del paquete junto a la posibilidad de renombrarlo además de informar el nombre del paquete padre.
- ***Árbol de contenidos del paquete***: A la izquierda, se encuentra un árbol donde se visualizan en forma de nodo las distintas partes de un paquete (acciones, objectModels, estados, validadores, subpaquetes y configuración de runtime) mientras que como hojas del mismo se presentan cada uno de los elementos en particular. Para el caso de la figura

3, se puede ver seleccionada la acción **`CDInformation`** además del objectModel **`cd`** y dos estados del paquete.

- ***Configuración del elemento seleccionado del árbol de contenido***: En el panel de la derecha de la ventana se ven las distintas posibilidades de configuración del elemento seleccionado en el árbol de contenidos de la derecha. Es decir, para el caso de la figura 3, donde en el árbol se encuentra seleccionada la acción **`CDInformation`**, se puede ver en este panel las posibilidades de configuración para dicha acción. Allí se ven campos de textos para el nombre identificador de la acción y para el nombre de la clase que realmente implementa la acción además de 5 solapas para todas las distintas configuraciones necesarias de una acción: object model de entrada y de salida, los estados particulares de la acción, la configuración de runtime y la validación. Para otros casos como el de definición de un objectModel o un estado propio del paquete se presentarán las configuraciones necesarias para cada uno.

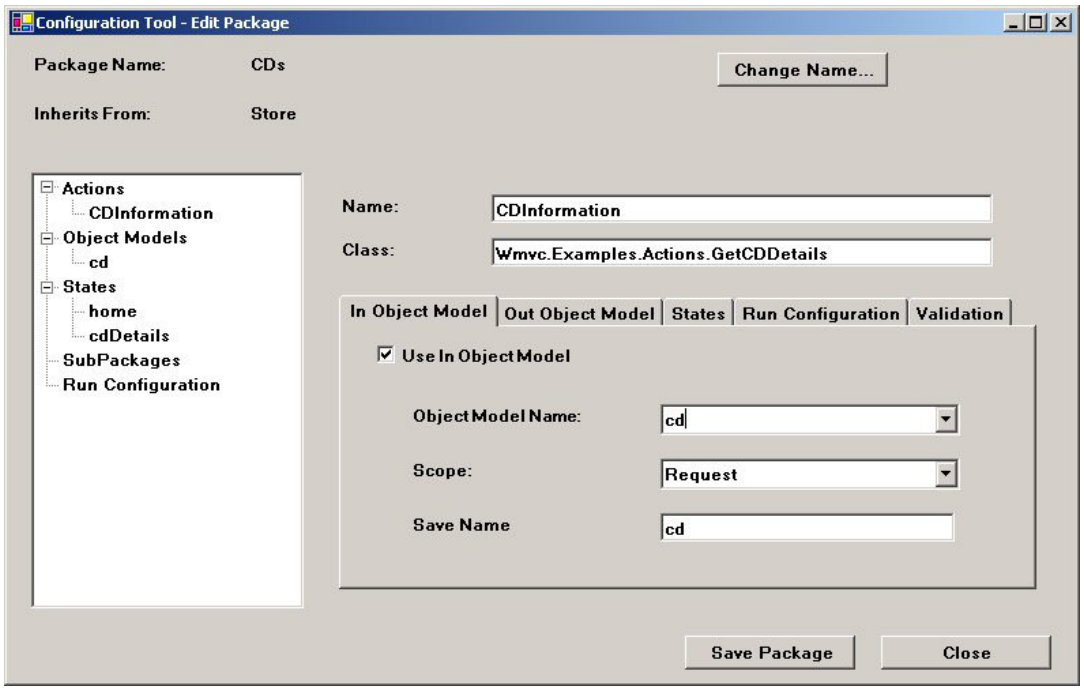


Figura 3: Edición de un paquete con el Configuration Tool

8.4 Source Code Writer

Esta aplicación facilita la tarea de los desarrolladores eximiéndolos de la escritura de la estructura de clases que extienden al framework De ésta manera, una vez definidos los archivos de

configuración, el desarrollador, mediante el uso de esta herramienta, puede generar todas las clases necesarias en forma automática.

Esta herramienta tiene como única funcionalidad la generación del código fuente de las clases indicadas en los archivos de configuración tanto para las acciones, como para los object models y los validadores de todos los paquetes.

Mediante la utilización de esta herramienta, los desarrolladores no necesitan ir recorriendo uno a uno, elemento por elemento los archivos de configuración generando las clases sino que alcanza con ejecutar el Source Code Writer y toda la generación de las mismas es realizada automáticamente. Esta herramienta no solo crea los archivos automáticamente sino que además documenta mínimamente el código fuente generado, indicando por ejemplo, para las acciones, bajo que nombre se instancia esta clase de acción y el tipo de Object Model que utiliza como entrada y salida.

8.5 Funcionamiento del "Source Code Writer"

El Source Code Writer posee una interfaz bastante parecida a la otra herramienta provista por el WMVC Framework explicada anteriormente y al igual que ésta se presenta en forma de aplicación de escritorio.

La aplicación se maneja con proyectos, los que son creados a partir de un archivo de configuración principal `wmvc.config` de cualquier aplicación. Una vez seleccionado éste, un archivo propio del Source Code Writer es creado con extensión `.scw` que no solo contiene la referencia al archivo de configuración principal inicialmente seleccionado sino que también almacena los valores básicos propios del programa de manera que el usuario del mismo no tenga que setearlos cada vez que el usuario necesite usar el programa.

Luego de cargado un proyecto, en la pantalla principal (figura 4), se puede ver lo siguiente:

- Árbol con la jerarquía de paquetes definidos en los archivos de configuración donde cada paquete tiene a su lado un checkbox que indica si será o no incluido al momento de generar el código.

- Un conjunto de posibilidades para chequear o deschequear más de un elemento del árbol a la vez con el objetivo de facilitar la selección de los paquetes en la jerarquía de paquetes definida para la aplicación.
- El panel de generación de código con el cual realmente se genera el código fuente. En dicho panel se configura el path base donde el código será creado junto a un set de opciones que permiten que la aplicación pueda generar el código fuente de solo una parte de la configuración o que ante la existencia de un archivo lo sobrescriba o lo deje tal cual lo encontró.

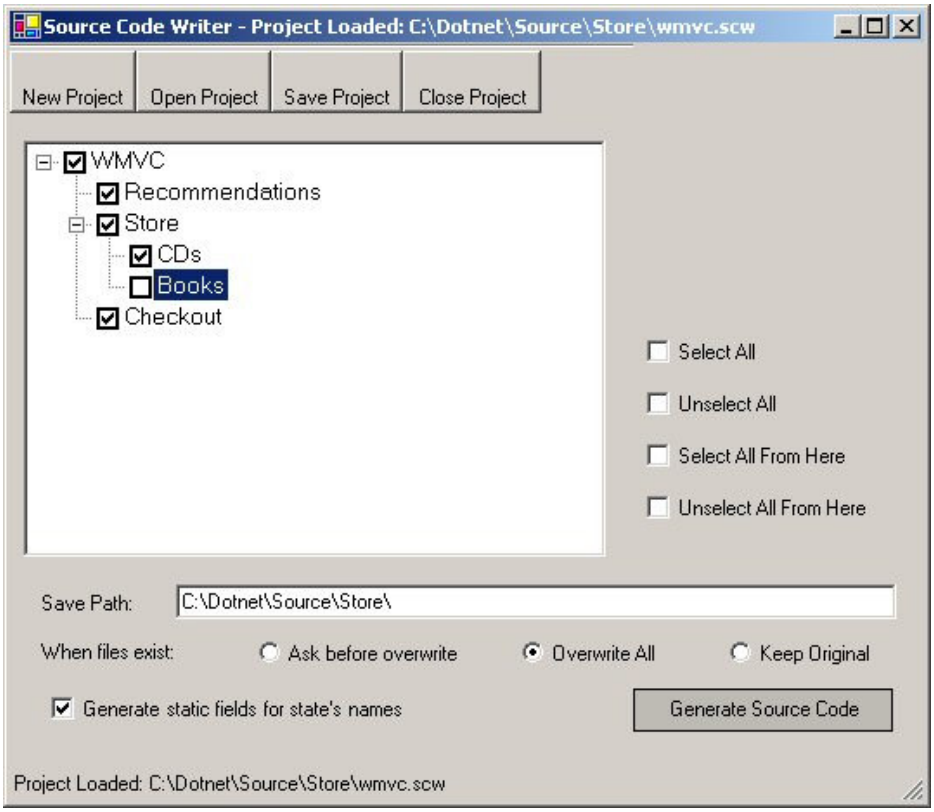


Figura 4 – Ventana principal del Source Code Writer

Una vez presentada en forma breve la aplicación pasaremos ahora a los pasos que se deben seguir para utilizar la aplicación y tener como resultado el código fuente de las clases mencionadas en los archivos de configuración.

Primero se debe cargar un proyecto, sea uno nuevo a través de la selección de un archivo **`wmvc.config`** o sea uno existente con extensión **`.scw`**. Luego se debe seleccionar del árbol de paquetes los paquetes para los cuales se desea generar el código fuente. Para selecciones múltiples se pueden utilizar las cuatro funciones ubicadas a la derecha.

Una vez finalizada la selección de paquetes se debe pasar entonces al panel inferior en el cual se encuentran las distintas posibilidades para generar el código fuente:

- ***SavePath***: indica el path o carpeta inicial donde se va a almacenar el código fuente.
- ***When files exist***: Esto plantea tres opciones posibles para elegir ante el caso que se corra el Source Code Writer una vez que todos o alguno de los archivos ya existan. Permite elegir entre guardar los originales, preguntar por cada uno o sobrescribir los ya existentes con los nuevos.
- ***Generate static fields***: Esto permite que para cada state posible para una acción genere una variable estática en la clase de la acción para no escribir en código los nombres de los estados.

Luego de finalizada la selección de los paquetes y de las distintas posibilidades para generar el código solo queda apretar el botón **"Generate Source Code"** para obtener los resultados esperados.

Capítulo 9: Conclusión y Trabajo Futuro

Como capítulo final de este trabajo, describiremos aquí algunas conclusiones sobre el trabajo realizado junto también a la enumeración de ciertas tareas a efectuar en un futuro cercano como forma de continuar con todo lo desarrollado en el transcurso de éste proyecto.

9.1 Conclusión

Como conclusión del trabajo realizado podemos pensar en los siguientes puntos:

- La implementación de un Framework MVC para el Framework Microsoft .NET, deja a la comunidad .NET un importante producto libre para su utilización y adaptación a las necesidades del desarrollador.
- Se provee una base importante para el desarrollo final de un Framework MVC que pueda ser constituido como producto en el mercado actual del software. Actualmente la implementación no se encuentra disponible como producto dado que no ha pasado por una etapa compleja de testeo ni tampoco de análisis y mejora de la performance.
- Aunque la implementación está orientada a ASP.NET, tanto para el desarrollo de aplicaciones web como mobile, gracias a la arquitectura definida para este framework WMVC, éste es extensible a:
    - Aplicaciones que utilicen el Framework de ASP.NET, ya sea para hacer uso de la web o de web para dispositivos móviles, pero que tengan comportamientos más específicos o ciertas restricciones que obliguen al desarrollador a no poder usar las estructuras básicas de las páginas del .NET.
    - Soportar cualquier otro tipo de vista ya sea por ejemplo XML, WinForms, servicios web, etc.
- La implementación del Framework WMVC provee a los desarrolladores que lo utilicen grandes posibilidades de extensión:
    - ***Controles Web***: Para los controles gráficos, tanto web como mobile, el WMVC Framework provee una interfaz de implementación que permite que cualquier control que implemente esa interfaz pueda utilizar para su gusto las propiedades especificas que estos controles WMVC agregan a los controles provistos por el .NET Framework.

- ***Niveles de Aislamiento***: Actualmente el Framework posee únicamente tres modelos de aislamiento diferente: (nada, proceso y ASP.NET). Sin embargo, la implementación de otros niveles de aislamiento para las acciones es algo completamente fácil y abierto a cualquier desarrollador.
- ***Modos de Ejecución***: Al igual que ocurría con los niveles de aislamiento, el WMVC Framework provee la posibilidad de implementar nuevos modos de ejecución de manera que el desarrollador pueda elegir e implementar uno particular de la forma en que el lo necesite.
- ***Diferentes Vistas***: La implementación de nuevas vistas para que sean soportadas por el WMVC Framework es algo completamente posible, ya que con la codificación de apenas un conjunto pequeño de clases se logra agregar una nueva vista al Framework.
- ***Diferentes comportamientos para vistas iguales***: Dentro de la posibilidad del punto anterior se encuentra también la posibilidad de realizar para una misma vista diferentes comportamientos como así también que dos tipos de vista diferente utilicen o se comporten de una forma similar.

9.2 Trabajo futuro

A continuación, se presentará una breve descripción de algunas tareas que representarán el trabajo a realizar a futuro. Algunas tareas de esta lista no pudieron ser realizadas durante el periodo de este trabajo por cuestiones tiempo y para que no se agrandara demasiado el proyecto sin tener resultados intermedios para mostrar. Por otro parte, otros puntos que aparecen en la lista fueron ciertas ideas que fueron apareciendo durante la construcción del Framework y otros durante el proceso de documentación. A su vez, el resto de los elementos son ideas a futuro que podrían mejorar el funcionamiento o la facilidad de uso del WMVC Framework o adaptarse a los cambios que se producen en su framework base, el Microsoft .NET Framework.

Continuar con ciertos detalles o mejoras en cuanto al desarrollo propio del framework como pueden ser los siguientes:

- ***La creación de wizards o asistentes***: Los wizards se podrían definir como un conjunto definido de acciones donde cada elemento del mismo es una acción. A su vez, en lugar de tener muchos estados resultantes de salida, las acciones de cada paso

del asistente tendrían como posibles estados resultantes a: siguiente, anterior, inicio, fin representando así los distintos caminos a seguir durante todo el asistente.

- ***Acciones compuestas***: Este tipo de acciones representaría a una colección de acciones que se ejecutan en forma similar a un pipeline. De esta forma una acción (lógica) en la vista podría generar más de una acción física en el modelo, logrando así una mayor modularización de los distintos pasos que deben ocurrir durante la ejecución de una acción.
- ***Forward desde la vista***: La creación de un control gráfico que permita efectuar un forward directo de una vista hacia la otra, simplificando y abstrayendo de esta forma el modo de redireccionamiento incondicional. Esto serviría específicamente para el desarrollo de menús estáticos y enlaces sin procesamiento entre páginas.
- ***Pipelines***: La investigación y desarrollo de diferentes tipos de pipelines permitiría ampliar infinitamente los modos de configuración y ejecución de las acciones, a la vez de cubrir mediante la utilización de estos pipelines varios de los puntos sugeridos como trabajo futuro en esta sección.
- ***Manejo de eventos***: Estudiar una mejora en la manera en que el desarrollador programa los eventos podría obtener como beneficio una más fácil utilización de los mismos en las vistas.
- ***Mejoras en performance***: Con el objetivo de adquirir una mayor eficiencia se podrían emplear métodos más eficientes y menos abiertos para la ejecución de las acciones. Entre ellos se encuentra la posibilidad de generar dinámicamente nuevos assemblies con código compilado donde el proceso de creación de objetos sea estático y tipado en lugar de utilizar reflexión y código "lento" como una búsqueda de datos en diferentes colecciones de objetos.

Proveer una completa integración del WMVC Framework a la principal herramienta de desarrollo para .NET como es el Visual Studio sería algo que favorecería enormemente a la utilización del framework. Esta integración se realizaría mediante un add-in que provee cierta funcionalidad como por ejemplo:

- El agregado de una barra de controles al toolbox ya existente en el Visual Studio. Así, el desarrollador podría sin tener que configurar nada, arrastras los controles WMVC web y mobile dentro de sus páginas como si fueran componentes originales del .NET Framework.

- Crear una solución WMVC, agregando un proyecto donde residan los object models, otro para el modelo de la aplicación, más uno diferente para las acciones. A su vez dentro de esa opción se podría configurar la/s vista/s a utilizar y que se creen proyectos propios para cada tipo de vista y se agreguen a la solución.
- Crear un proyecto WMVC especifico, sea para acciones, object models, modelo o hasta el de una aplicación web o mobile WMVC instanciando el proyecto con los archivos y templates que estos necesitan.
- Crear una acción u object model WMVC, escribiendo automáticamente un template adecuado para ello.
- Crear una nueva WMVC Web Page o WMVC Mobile Page agregando en cada caso los templates y elementos necesarios para su puesta en marcha.

Por otro lado, también se plantea como trabajo futuro la posibilidad de hacer que este Framework WMVC sea un producto utilizable por la comunidad de desarrolladores .NET para lo cual serían necesario una traducción, no necesariamente completa de este documento así como también una pequeña publicidad mediante la escritura de artículos en los distintos sitios sobre desarrollo para la comunidad de .NET, como pueden ser Code Project (www.codeproject.com), GotDotNet (www.gotdotnet.com), ASP.NET (www.asp.net) y DevX (www.devx.com).

Por último, queda pendiente la adaptación del Framework WMVC a la nueva versión del Microsoft .NET Framework: versión 2.0 a salir próximamente. Actualmente el Framework .NET se encuentra en su versión 2.0 Beta, aunque ésta ha sufrido enormes cambios desde la versión Alfa a la actual principalmente donde el WMVC Framework más lo sufre: en el subconjunto de clases que manejan las aplicaciones web y mobile. Debido a ello, se dejo para un futuro la adaptación del WMVC Framework a la nueva versión del Framework .NET.

Referencias

[1] The Common Gateway Interface - http://www.w3.org/CGI/

[2] WikiPedia - http://en.wikipedia.org/wiki/Common_Gateway_Interface

[3] Webopedia - http://www.webopedia.com/TERM/C/CGI.html

[4] National Center for Supercomputing Applications at the University of Illinois at Urbana - Champaign, IL - http://hoohoo.ncsa.uiuc.edu/cgi/overview.html

[5] An introduction to PHP – http://www.devshed.com/c/a/PHP/An-Introduction-to-PHP/

[6] Official PHP Online Documentation - http://www.php.net/docs.php

[7] http://en.wikipedia.org/wiki/PHP

[8] WikiPedia - http://en.wikipedia.org/wiki/Active_Server_Pages

[9] Microsoft Official Active Server Pages Site - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/activeservpages.asp

[10] Documentación de Java/Servlet - http://java.sun.com/products/servlet/

[11] Documentación de Java/JSP - http://java.sun.com/products/jsp/

[12] The Evolution of JSP - http://www.macromedia.com/devnet/server_archive/articles/evolution_of_jsp.html

[13] A system of Patterns - Pattern Oriented Software Architecture Volume 1 – Buschmann, Meunier, Rohnert, Sornmerlad, Stal

[14] MVC desde Smalltalk a la Web: Análisis de la Arquitectura que produjo un cambio en el diseño de aplicaciones – María Alejandra Gos

[15] Alan Knight, Dai, "Objects and the Web", IEEE Software, March/April 2002

[16] Design Patterns Elements of reusable object-oriented software,
E. Gamma, R. Helm, R. Johnson, J. Vlissides, Addison Wesley 1995.

[17] Documentación de Struts - http://jakarta.apache.org/struts/

[18] Documentación de MSDN para el .NET Framework
http://msdn.microsoft.com/netframework/

[19] Documentación de MSDN para ASP.NET
http://msdn.microsoft.com/asp.net/

Apéndice A: Configuración del WMVC Framework

A.1 Introducción a la configuración del WMVC Framework

El WMVC Framework es configurable mediante un conjunto de archivos con formato XML que se deberán crear en la carpeta **"ConfigFiles"** de la aplicación web. Es decir, por ejemplo si la aplicación web ocupa el directorio

```
C:\inetpub\wwwroot\miAplicación
```

entonces el directorio de configuración debería estar ubicado en:

```
C:\inetpub\wwwroot\miAplicacion\configFiles
```

El archivo de configuración principal que debe poseer toda aplicación web es el **"WMVC.config"**. Tanto este archivo como los archivos de configuración que se desprendan de él se encuentran divididos en ocho partes:

- ***Especificación del propio paquete***: Aquí se declara información general para todo el paquete y que es directamente heredada por cada acción.
- ***Especificación de aislamiento***: En esta sección se define el tipo de aislamiento que tienen las acciones del paquete.
- ***Declaración de acciones***: En esta parte se describen y definen las acciones del paquete propietario del archivo de configuración.
- ***Declaración de ObjectModels***: En esta parte del archivo se declaran todos los objectModels del paquete. Estos objectModels tienen la misma regla de accesibilidad que los estados: son públicos para el paquete y para cualquier subpaquete del paquete que contiene la definición.
- ***Declaración de validadores***: En esta región se definen los distintos validadores propios del paquete.
- ***Declaración de los estados***: Aquí, al igual que en la sección de las acciones, se declaran los estados a los cuales se podrá llegar luego de ejecutar cualquier acción que esté definida o en el propio paquete o en un cualquier subpaquete, no necesariamente directo.

- ***Declaración de subpaquetes***: En ella se enumeran los nombres de los paquetes y qué archivo de configuración tiene cada paquete. Esto permite simular una jerarquía de paquetes similar a la de los paquetes del lenguaje Java o .NET con el objetivo de permitir la división de la aplicación en diferentes paquetes.
- ***Declaración del conjunto de Dlls a usar***: Aquí se enumera el conjunto de dlls que necesita usar el WMVC Framework al momento de ejecutar las acciones.

Su estructura principal sería como se puede apreciar en la figura 1.

```
<?xml version="1.0" encoding="utf-8" ?>
- <Package name="_____">
    <!--  == Isolation information ==  -->
    <Isolation />
    <!--  == Actions definition for the current package ==  -->
  - <Actions>
      <Action>...</Action>
    </Actions>
    <!--  == ObjectModels definition for the current package  ==  -->
  - <ObjectModels>
      <ObjectModel>...</ObjectModel>
    </ObjectModels>
    <!--  == Validators definition for the current package  ==  -->
  - <Validators>
      <Validator>...</Validator>
    </Validators>
    <!--  == States definition for the current package  ==  -->
  - <States>
      <State>...</State>
    </States>
    <!--  == SubPackages definition for the current package  ==  -->
  - <Packages>
      <Package>...</Package>
    </Packages>
    <!--  == Dlls definition for the current package  ==  -->
  - <Dlls>
      <Dll>...</Dll>
    </Dlls>
</Package>
```

Figura 1: Estructura principal de un archivo de configuración

A.2 Definición de cada una de las partes de un archivo de configuración

Una vez que ya hemos presentado la estructura general de los archivos de configuración pasaremos ahora a ver un poco más en detalle que es cada una de las 7 partes del archivo y que permite configurar cada una.

Para hacer esto, supondremos que necesitamos configurar una aplicación web de negocios que se dedica a la venta de discos compacto. Este sistema necesita que el usuario se loguee al mismo para poder navegar y las compras se efectúan a través de cierta información personal que el usuario haya cargado, como puede ser tu tarjeta de crédito, su domicilio y su dirección de correo electrónico. .

A.2.1 Especificación del propio paquete

Como ya se mencionó previamente, el WMVC Framework permite dividir la configuración de la aplicación en paquetes. Por lo tanto, explicaremos aquí cómo se define un paquete en el WMVC Framework.

Lo primero que se debe hacer para definir un paquete nuevo es agregar una línea en el archivo **`.config`** de configuración del paquete donde vaya a colgar el mismo (ver A.2.7). Esto es válido para todos los paquetes excepto para el paquete principal del sistema, el cual tiene por nombre "WMVC" y se encuentra definido en el archivo **`wmvc.config`**.

Para el ejemplo presentado en la sección A.2, necesitaríamos definir tres paquetes distintos: uno que defina toda la información y comportamiento del usuario, otro la funcionalidad del carrito de compras y el checkout y otro que obtenga datos de los discos compactos que se desean vender. El paquete de información del usuario lo dejaremos como parte del paquete principal mientras que el que maneja el checkout y el que maneja información de los CDs serán subpaquetes del primero.

```
<Package name="cdStore" actionMode="OnRequest">
    ...
</Package>
```

Figura 2: Definición del encabezado del paquete

En la figura 2 se puede ver como se declara el encabezado del paquete CdStore en el archivo **`CdStore.config`**. Allí se puede advertir también que en la misma declaración del paquete se puede setear información sobre el modo en que se van a ejecutar las acciones (**`onPostback`**, **`onRequest`**). Este valor no es obligatorio ya que puede heredar el especificado por su paquete padre, pero si es definido entonces éste valdrá para todas las acciones de este paquete y de sus subpaquetes que no redefinan esta propiedad.

A.2.2 Especificación de Aislamiento

Cada acción puede definir un nivel de aislamiento distinto para su ejecución, pero dado que generalmente el mismo nivel es definido para todas las acciones o al menos para las de un mismo paquete debido a su relación en cuanto a la funcionalidad, entonces el WMVC Framework permite también setear el nivel de aislamiento en el paquete. Además vale la misma regla que con el **`ActionMode`** por lo que se permite que no sea necesario configurar el nivel de aislamiento por cada acción sino por paquete y que las acciones y subpaquetes hereden esa configuración.

Actualmente el Framework provee tres niveles de aislamiento distintos de los que, uno (**`NONE`**) no necesita de ninguna configuración mientras que los otros dos (**`ASPNET, PROCESS`**) deben especificar la dirección IP, el puerto, el path interno y el protocolo que deben usar para comunicarse con el servidor donde se encuentra el modelo de negocios de la aplicación.

```
<Isolation
    level="ASPNET" Protocol="IP"
    IP="192.168.0.182" Port="80" Path="cdStoreModel"
    >
</Isolation>
```

Figura 3: Definición de aislamiento del paquete

En la figura 3, se muestra un ejemplo donde las acciones que tengan esta configuración se ejecutarán en la máquina con IP 192.168.0.182 dentro del proceso ASPNET, en el directorio virtual cdStoreModel y la comunicación se realizará usando el protocolo IP por el puerto 80.

A.2.3 Definición de las acciones

Las acciones son por lejos los elementos más complejos de configurar, no porque sean difíciles sino por las múltiples combinaciones que se permiten entre sus propiedades. Cada acción permite directamente en su nodo configurar:

- ***Name***: Es uno de los dos campos obligatorios y representa el nombre que identificará esta acción en la aplicación.
- ***ActionClass***: Indica la clase que se deberá instanciar para ejecutar la acción. Es el otro campo obligatorio.

- ***ActionMethod***: Nombre del método del ActionClass que se debe llamar para ejecutar la acción. Este campo permitiría disminuir la cantidad de clases Action insertando más de una acción por clase.
- ***ActionMode***: Modo en que se ejecuta la acción (**`onRequest`**, **`onPostback`**)
- ***ServerSideValidation***: Nombre del Validador a usar para esta acción.

Como nodos internos al de la acción se pueden definir opcionalmente los siguientes cuatro elementos: ObjectModel de entrada, ObjectModel de salida, aislamiento de la acción y estados posibles de la ejecución.

El ObjectModel de entrada se define con **`<inObjectModel>`** y posee información sobre el objectModel que se enviará como Request para la ejecución de la acción mientras que el ObjectModel de salida demuestra el ObjectModel resultante de la ejecución y su configuración se encuentra en **`<outObjectModel>`**. Para ambas configuraciones de objectModel se indican tres propiedades:

- ***Name***: Indica el nombre del ObjectModel
- ***Scope***: Indica el alcance desde donde se debe obtener o salvar la instancia de ObjectModel.
- ***SaveName***: contiene el nombre con el que es accedido en el scope indicado antes.

Para configurar el nivel de aislamiento de la acción se aplica lo explicado en A.2.2 y para definir los estados resultantes de la ejecución de una acción se cumple con lo que se explicará luego, en la sección: “Definición de los estados y sus vistas” (A.2.6)

```
<Action name="newUser" actionMode="onRequest"
        serverSideValidation="verifyUser"
        actionClass="WMVC.Examples.BookStore.Actions.NewUser">
    <InObjectModel scope="Session" name="UserInfo" saveName="user"/>
    <States>
        ...
    </States>
</Action>
```

Figura 4: Definición de una acción

En la figura 4, se distingue una definición de acción llamada newUser y que representa a la clase definida en ActionClass. Define un validador para verificar cierta información previa a la acción y además indica explícitamente que el modo de ejecución de la acción será **`onRequest`**. Por

último configura la utilización de un objectModel como entrada que será obtenido desde la sesión mediante la clase indicada en saveName.

A.2.4 Definición de los objectModels

En esta sección se configuran todos los objectModels usados por las acciones. Es importante recordar que los objectModels pueden ser usados por varias acciones, y no solamente por una. De esta manera, uno podría definir un objectModel solo para realizar algún comando que lleve dos o más pasos.

La definición de cada objectModel es fácil ya que solo se debe definir un nombre e indicar el nombre de la clase que se debe instanciar cuando alguna acción se refiera a éste. Entonces los campos son:

- ***Name***: nombre del objectModel.
- ***Type***: nombre de la clase del objectModel que se debe instanciar. Esta clase debe implementar la interfaz vacía **`WMVC.IObjectModel`**.

```
<ObjectModels>
    <ObjectModel name="UserInfo"
        type="WMVC.Examples.BookStore.ObjectModels.UserInformation" />
    <ObjectModel name="Cart"
        type="WMVC.Examples.BookStore.ObjectModels.Cart" />
</ObjectModels>
```

Figura 5: Definición de un ObjectModel

Volviendo a la aplicación de venta de CDs en la figura 5 se ve la definición de dos objectModels de manera de mostrar con un ejemplo la explicación anterior.

A.2.5 Definición de los validadores

Los validadores del framework son definidos en la sección Validators de los archivos de configuración. Para cada validador en particular solo se deben configurar dos propiedades:

- ***Name***: Nombre que representa al validador en la aplicación.
- ***Type***: Clase que se debe instanciar para ejecutar el validador. Esta clase debe heredar de **`WMVC.Validator`**.

```
<Validators>
    <Validator name="verifyUser"
            type="WMVC.Examples.BookStore.Validators.VerifyUser" />
</Validators>
```

Figura 6: Definición de un Validador

En la figura 6 se muestra un ejemplo simple, donde se define el validador verifyUser el cual se encuentra implementado en la clase indicada en el atributo type.

A.2.6 Definición de los estados y sus vistas

En la sección de configuración de States se definen alias a las páginas para que desde la aplicación no se escriba la dirección de la ventana o página siguiente que se debe cargar. Estos estados serán usados generalmente, luego de ejecutar una acción ya que el último paso de una acción es redireccionar mediante el uso de un estado a la ventana o página siguiente.

Los estados se pueden configurar tanto en un paquete, como dentro de una acción. El alcance que tiene una definición de estado varía dependiendo en donde se lo definió. Si fue definido en un paquete, cualquier acción de ese mismo paquete o de cualquier subpaquete lo puede utilizar mientras que un estado definido dentro de una acción tiene como único alcance a la misma acción.

Para cada State se debe definir:

- ***Name***: Nombre que se va a utilizar desde el código fuente para referirse al este estado.
- ***Type***: El type permite definir el tipo de estado al que se refiere. Las posibilidades son: **RV** (Forward), **EX** (Exception). Este atributo solo sirve para sea más fácil entender las acciones y sus resultados.
- ***Default***: Es opcional y solo debe haber uno por definición de estados. Este atributo indica que si ningún estado es seteado al finalizar una acción, entonces si este estado se encuentra en el alcance de la acción, será automáticamente asignado como valor resultado.
- ***Views***: Las views es una colección de elementos view que definen para cada tipo de requerimiento a donde se debe redireccionar.

En cuanto a la definición de las vistas para cada una se debe especificar:

- ***type***: Aquí se define si el requerimiento es "mobile", "web" o "*" que vendría a representar a todos a la vez.
- ***path***: Se indica el path al cual debe redireccionar la aplicación.

```
<States>
    <State name="viewCD" type="RV" >
        <Views>
            <View type="web" path="CDStore/viewCDDetails.aspx" />
            <View type="mobile" path="CDStore/viewCD.aspx" />
        </Views>
    </State>
    <State name="CDNotFound" type="EX" >
        <Views>
            <View type="*" path="error.aspx" />
        </Views>
    </State>
</States>
```

Figura 7: Definición de un estado y sus vistas

Volviendo al ejemplo del comercio electrónico y más precisamente a la acción viewCD definida en el paquete CDStore, de la cual sus estados pueden ser vistos en la figura 7. Allí se definen dos estados locales a la acción. En el primero, viewCD, cuando es utilizado redirecciona a la página "CdStore/viewCdDetails.aspx" en caso que el pedido sea web y a "CdStore/viewCd.aspx" en el caso que sea mobile. Por otra parte, también se declara otro estado, definido como de excepción, llamado "CDNotFound" que cuando se utilice no importa que tipo de pedido se realice siempre ira a "error.aspx".

A.2.7 Definición de los subpaquetes

En la sección de Subpaquetes se definen todos los subpaquetes del paquete actual. Para cada paquete simplemente se debe definir el nombre del mismo dado que la ubicación de su archivo de configuración es obtenida automáticamente a partir del paquete actual.

```
<Subpackages>
    <Subpackage name="Checkout" />
    <Subpackage name="CdStore" />
</Subpackages>
```

Figura 8: Definición de subpaquetes

Es decir, para el ejemplo que muestra la figura 8, en cualquiera de las dos definiciones de subpaquetes ocurre lo mismo. Dado que son subpaquetes del paquete principal, el nombre de sus

archivos de configuración son directamente **CheckOut.config** y **CdStore.config** pero para por ejemplo un paquete cualquiera que sea definido como subpaquete de CdStore, su archivo de configuración deberá ser **CdStore.nombreDelPaquete.config**.

A.2.8 Definición de las dlls a utilizar

En esta sección del archivo de configuración se listan todas las librerías que se usarán en la aplicación tales como las que contienen la definición de acciones, las librerías donde estén definidos los objectModels y opcionalmente cualquier otra clase que el WMVC Framework deba instanciar para el funcionamiento del controlador. Dos puntos importantes por considerar son:

- No es necesario listar entre las librerías el archivo dll del WMVC Framework.
- Las Dlls deberían configurarse en el paquete principal (archivo **WMVC.config**). De todas maneras no está mal que se definan en otro lado, ya que no importa donde sean configuradas su alcance es cualquier paquete de la aplicación. La recomendación de que se definan en el paquete principal es solamente por claridad dado que ese paquete es el único paquete que define elementos visibles en todo el sistema.

```
<Dlls>
    <Dll path="D:\WMVC\Examples\CdStoreModel.dll" />
    <Dll path="D:\WMVC\Examples\CdStoreModel.Actions.dll" />
    <Dll path="D:\WMVC\Examples\CdStoreModel.ObjectModels.dll" />
    <Dll path="D:\WMVC\Examples\CdStoreModel.Validators.dll" />
</Dlls>
```

Figura 9: Definición de dlls

La definición de una dll en el archivo de configuración es extremadamente fácil ya que solo involucra un atributo ***Path*** el cual indica la ubicación del archivo dll. En la figura 9 se puede ver un ejemplo claro de la configuración de Dlls.

A.3 Modificaciones en los archivos de configuración

¿Qué sucede ante una modificación de un archivo de configuración? Aunque actualmente esta funcionalidad no se encuentra todavía implementada se tienen pensadas dos posibles soluciones.

Una de las soluciones sería que la persona que realizó el cambio en la configuración deba ejecutar un pequeño programita que le avise al controlador de la aplicación que su configuración

sobre la cual trabaja ha sido cambiada y que, por lo tanto, debe atender las nuevas peticiones obteniendo las configuraciones leyendo los archivos del disco tal cual lo hizo apenas se inició el sistema.

Este pequeño programita sería accesible desde línea de comando y además desde el programa de configuración "Configuration Tool". De esta manera el Configuration Tool proveerá todas las herramientas necesarias para crear y modificar los archivos de configuración permitiendo al usuario abstraerse del formato de los mismos y trabajar únicamente con esta herramienta para configurar la aplicación.

La otra posibilidad para el problema de la modificación de los archivos en tiempo de ejecución sería que la propia instancia de `ConfigurationManager` verifique las actualizaciones de sus archivos de configuración y ante un cambio de éstos, reaccione recargando la configuración de los mismos.

A.4 Ventajas de los archivos de configuración

Si bien la información sobre la configuración del WMVC Framework se encuentra en los archivos de configuración, ésta información podría haberse definido en múltiples lugares. Sin embargo, luego de realizar un análisis de las posibilidades se concluyó que los archivos de configuración era la posibilidad más viable debido a que contaba con ciertas ventajas por sobre las demás. En lo sucesivo se enumerarán las diferentes posibilidades y se analizará brevemente las ventajas y desventajas de cada una de ellas.

A.4.1 Archivos de Configuración

Esta posibilidad, que fue la finalmente elegida, era por sobre todas las demás la que contaba con mas ventajas. Entre ellas se encontraban:

- Facilidad de programación del módulo de lectura y escritura de los archivos de configuración dado que .NET provee todas las herramientas para el manejo de archivos xml.
- Facilidad en la lectura de la configuración tanto por el ser humano como por un programa dado que estos archivos tenían todas las ventajas de los archivos XML.

- La configuración del sistema se encontraba toda junta en una carpeta (configFiles) lo que daba notable claridad para el manejo y control de la misma.
- 100% compatible con las configuraciones estándares definidas por el .NET Framework en el cual la mayoría de las configuraciones son provistas por archivos con formato XML y extensión `.config`.
- Provee una gran posibilidad de extensión de manera que nuevos Frameworks basados en el WMVC Framework puedan también basar sus configuraciones en los mismos archivos de configuración.
- Las actualizaciones de la configuración durante el desarrollo consiste simplemente en la modificación del archivo del paquete que se desea cambiar.
- Permite actualizaciones de la configuración en tiempo de ejecución. Esto se hace implementando cualquiera de las dos posibilidades mencionadas en A.3.2.

A.4.2 Configuración mediante atributos del .NET

El concepto de atributo provisto por el .NET Framework permite agregar metadata de manera que esta pueda ser accedida tanto sea con una herramienta post-compilación como más tarde, en tiempo de ejecución.

Estos atributos podrían haber sido utilizados para definir la configuración del WMVC Framework. Por ejemplo, en el código fuente de una acción se deberían definir atributos que indiquen el nombre de la acción, los objectModels que utiliza, los estados que define, etc.

Sin embargo, a pesar de ser una de las posibilidades utilizadas en la programación .NET, esta implementación tenía ciertas desventajas tales como:

- Los atributos son poco legibles cuando deben contener mucha información como era el caso de las acciones ya que para cada una de ellas se debía definir mucha meta-información.
- La configuración se encontraba separada a través de todas las clases de la aplicación por lo que tener una vista general de la misma era directamente imposible.
- Para efectuar modificaciones, en tiempo de desarrollo, no solo se debe modificar el archivo de código fuente que se desee sino también, posiblemente algún otro archivo de

código fuente que dependa de la modificación además de la consecuente necesidad de recompilar el módulo modificado.

- Las modificaciones en la configuración en tiempo de ejecución son directamente irrealizables dado que la necesidad de recompilar el código fuente llevan a cambios en los archivos funcionales de la aplicación.

A.4.3 Herramienta post-compilación

Otra posibilidad que se planteó fue la de desarrollar una herramienta de post-compilación que, a partir del código fuente del sistema y de la utilización de algún atributo o archivo de configuración pueda generar la información de configuración necesaria. Esto fue desechado casi inmediatamente debido a que no solo tenía los mismos problemas que la configuración mediante la utilización de atributos (A.4.2) sino que además sumaba otros problemas como:

- La configuración que el programador debía utilizar nunca era definida explícitamente por él, por lo que hacía que éste la desconozca o no la conozca en su totalidad.
- La utilización de la herramienta de post-compilación hace más lento el proceso de desarrollo y podría generar problemas de incompatibilidad con algún otro Framework que se utilice en post-compilación como podría ser alguna capa de persistencia.
- La herramienta de post-compilación no era completa por lo que el programador de todas maneras debía definir parte de la configuración.

A.4.4 Configuración definida directamente en el código fuente de la aplicación

La configuración definida directamente en código fuente del lenguaje también fue una posibilidad que enseguida fue descartada debido a que, no solo contaba con muchas de las desventajas mencionadas para la definición con atributos sino que también agrega dos desventajas importantes: el desarrollador debe programar toda la metadata y además no se provee una buena separación entre los distintos componentes provistos por el WMVC Framework.

Por ejemplo, para el caso de las acciones se debería definir la clase que hereda de `WMVC.Action` pero en lugar de nada más tener que implementar el método de ejecución también debería implementar métodos para obtener el nombre de la acción, obtener los nombres de los objectModel de entrada y Salida, obtener los estados a utilizar, etc.

Esto no solo agrega responsabilidad al desarrollador, ya que debe realizar el la programación de la configuración, sino que por sobre todas las cosas, en lugar de acelerar los tiempos de desarrollo, que es uno de los objetivos del Framework, los retrasa. Esto se debe a que al tener la aumentar la cantidad de código, dado que ahora también debe codificar la configuración, aumenta proporcionalmente la cantidad de posibles errores.